

The Targeting Expert

When doing targeting there is a lot of information to be provided, for instance how to generate the C source code, how to configure the SDL to C compiler, which compiler to be used etc.

The Targeting Expert is a tool managing the complete process of targeting (for the Cadvanced and the Cmicro SDL to C compilers only).

The easiest way to get an executable from an SDL specification (just generate C code for a complete SDL system - compile and link the executable) is as well supported as more complex cases. Therefore there are several Pre-defined Integration Settings which are ready-to-use.

Furthermore the Targeting Expert is able to re-use the deployments done in the Deployment Editor (see “The Deployment Editor” on page 1705 in chapter 41, *The Deployment Editor*) and to configure any single option thinkable in the process of doing targeting with Telelogic Tau. Those options can be the SDL to C compiler’s options, the compiler, linker and make tool options, for example.

After all this chapter is a description of the Targeting Expert Interactive Mode and a reference for the Batch Mode.

A list of FAQs (frequently asked questions) can be found at the end of this document.

Introduction

The Targeting Expert assists you in setting up and configuring a complete target application. A sub-set of the Targeting Expert functionality can be used in [Batch Mode](#) when it is only desired to re-build the complete system (or just several components).

Starting the Targeting Expert

In graphical Mode

Starting the Targeting Expert in the graphical mode means that the user interface is visible and you can directly interact by selecting function and menu items.

You enter the graphical mode by selecting *Targeting Expert* in the Organizer's *Generate* menu or by using the following command line options:

```
sdttax [ -h | -v | -pdm <partition_diagram>.pdm | -sdt <system-
name>.sdt | -pr <systemname>.pr ] [ -t <target> ] [ -hostsim | -re-
alsim | -sim | -val | -makeall ]
```

Option	Relevance
-v	The version number is shown and the application exits.
-h	Information about the usage is shown and the application exits.
-pdm <partition_diagram>.pdm	The partitioning diagram file to be used.
-sdt <systemname>.sdt	The system file to be used.
-pr <systemname>.pr	The system's PR file to be used
-t <target>	The qualifier of the SDL system's part to build
-makeint <integration>	Make the given integration
-makeall	Re-make last configuration

More information about the user interface can be found in "[The Main Window](#)" on [page 2830](#).

Introduction

See [“Interactive Mode” on page 2836](#) to get familiar with the Targeting Expert functionality.

In Batch Mode

The command line mode is also called [Batch Mode](#). Please see the appropriate sections for a more detailed description.

Possible command line options are:

taexbatch [-v | -h | -yes | -no]

Option	Relevance
-v	The version number is printed to stdout and the application exits.
-h	Information about the usage is printed to stdout and the application exits.
-yes	All the questions that probably come up will be answered with 'yes'.
-no	All the questions that probably come up will be answered with 'no'.

For more information about the commands allowed in a batch file see [“Description of Batch Mode Commands” on page 2887](#).

The Graphical User Interface

This section describes the appearance and functionality of the graphical user interface of the Targeting Expert (sdttax). Some user interface descriptions common to all tools in Telelogic Tau can be found in [chapter 1, *User Interface and Basic Operations*](#).

The Main Window

When you start the Targeting Expert in the interactive mode its main window is displayed.

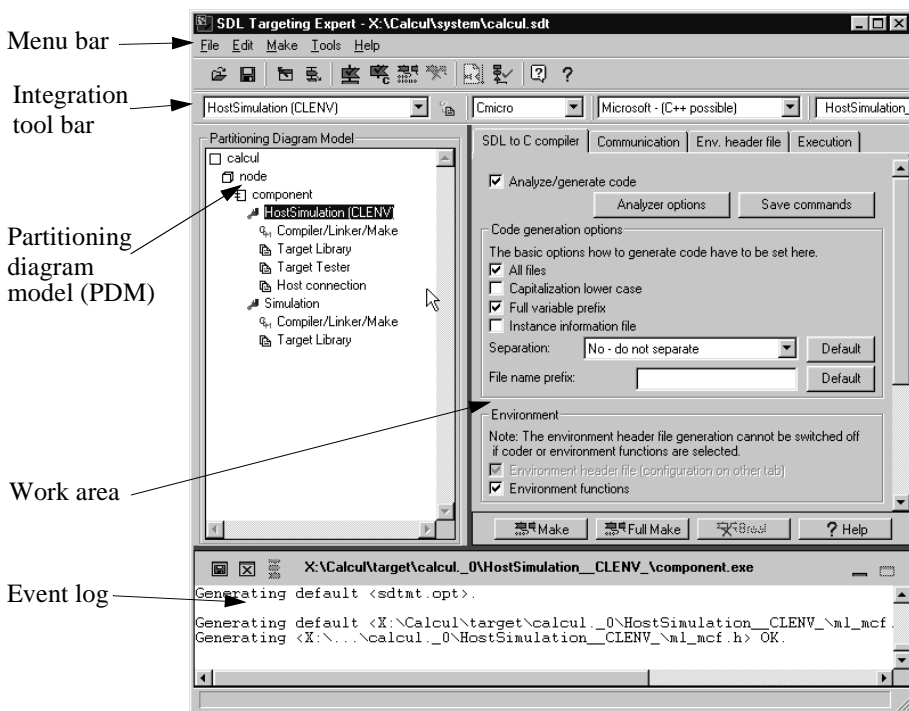


Figure 505: The Targeting Expert main window

The Menu Bar

This section describes the menu bar of the Targeting Expert Main window and all the available menu choices.

The menu bar contains the following menus:

- *File Menu*
- *Edit Menu*
- *Make Menu*
- *Tools Menu*
- *Help Menu*

(See “Help Menu” on page 15 in chapter 1, *User Interface and Basic Operations*.)

Configurable Menus

In Telelogic Tau, some menu choices may be available through the concept of user-defined menus. For more information, see “Defining Menus in the SDL Suite” on page 18 in chapter 1, *User Interface and Basic Operations*.

The definition file for user-defined menus searched for by the Targeting Expert is called `taex-menus.ini`.

There are several placeholders possible to be used with the `FormattedCommand` clause in `taex-menus.ini`.

placeholder	... will be replaced by ...
%s	system directory
%b	target directory (the one given in the Organizer)
%t	sub target directory (the one calculated depending on the selected component and integration)
%e	executable name (inclusive extension)
%i	intermediate directory

File Menu

The *File* menu contains the following menu choices:

- *Open*

- Save
- Exit

See “File Menu” on page 8 in chapter 1, *User Interface and Basic Operations*.

Edit Menu

The *Edit* menu gives access to the enlargement of the Targeting Expert configuration files.

- Add Compiler
- Edit Compiler Section
- Remove Compiler
- Add Communications Link
- Remove Communications Link
- *Edit Makefile*

A text editor is displayed where you can modify the generated makefile.

- *Edit Configuration Header File*

A text editor is displayed where you can modify the generated configuration header file.

- *Edit Environment File*

A text editor is displayed where you can modify the generated environment C file.

Make Menu

The *Make* menu allows to control the make process of the Targeting Expert.

- *Analyze*

The selected integration will be analyzed.

- *Generate code*

C code will be generated for the selected integration.

- *Make all*

All the configured components will be made again.

- *Clean*

All the object files in the object directory will be removed.

Tools Menu

The *Tools* menu gives access to other Telelogic Tau tools.

- *Show Organizer*

The Organizer main window is displayed.

- *Load SDL System*

The partitioned SDL system that is currently worked on in the Targeting Expert can be loaded into the Organizer. Use this entry if the Targeting Expert main window is displayed after the execution in the batch mode has failed.

- *Utilities > DOS to UNIX*

It is possible to specify a list of ASCII files. All the files in this list will be converted from DOS to UNIX style. See [“DOS to UNIX” on page 2935](#)

- *Utilities > UNIX to DOS*

It is possible to specify a list of ASCII files. All the files in this list will be converted from UNIX to DOS style. See [“UNIX to DOS” on page 2935](#)

- *Utilities > Indent*

The indentation of all the ASCII files in the list of files specified by the user will be corrected. See [“Indent” on page 2935](#)

- *Utilities > Preprocess*

The list of generated C files that is specified by the user will be pre-processes. See [“Preprocessor” on page 2936](#)

- *Wizards > TCP/IP communication*

A wizard dialog pops up and allows you to configure the TCP/IP communication between different components. See [“TCP/IP signal sending” on page 2877](#)

- *SDL > Simulator UI*

This menu choice starts a new, empty Simulator UI. Several Simulator UIs may exist at the same time. See “Graphical User Interface” on page 2130 in chapter 50, The SDL Simulator.

- *SDL > Validator UI*

This menu choice starts a new, empty Validator UI. Several Validator UIs may exist at the same time. See “Graphical User Interface” on page 2281 in chapter 53, The SDL Validator.

- *SDL > SDL Target Tester*

This menu choice starts a new, empty SDL Target Tester UI. See “Graphical User Interface” on page 3562.

- *Customize*

A dialog pops up and allows to customize the Targeting Expert. See “Customization” on page 2850 for more details.

The Integration Tool Bar

The integration tool bar should be used to (from left to right)

- Select the Pre-defined Integration Settings
- Get help about the selected integration
- Select an SDL to C Compiler
- Select a C Compiler to be used

Furthermore it should also be used to handle pre-defined and user-defined integration settings (see “Handling of Settings” on page 2846), i.e.

- to set a new user settings file name
- to export user settings as pre-defined settings
- to import node or application settings

The Work Area

The work area of the Targeting Expert is used for giving input masks for the different configurations and scalings.

The different input masks are described in

- “Configure how to Make the Component” on page 2874
- “Configure Compiler, Linker and Make” on page 2856
- “Configure and Scale the Target Library” on page 2872

- [“Configure the SDL Target Tester \(Cmicro only\)” on page 2873](#)
- [“Configure the Host \(Cmicro only\)” on page 2874](#)

The Event Log

The Targeting Expert has an event log containing information about found inconsistencies, read/write problems concerning file access, output, etc.

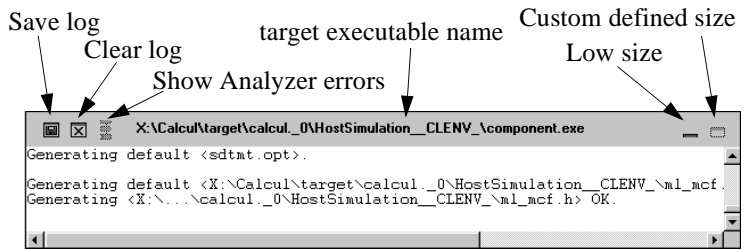


Figure 506: The event log

Analyzer/Compiler Errors

You can access analyzer and compiler errors by double clicking the error message in the event log.

- In case of an analyzer error the SDL Editor is displayed with the erroneous SDL symbol highlighted.
- In case of a compiler error the selected editor is displayed with the erroneous line highlighted.

Note 1: the selected editor must be able to highlight the desired line. It has to be selected in the [Customization](#) of the Targeting Expert.

Note 2: the compiler's error message syntax must be described in the [Compiler Error Descriptions](#) section in file `sdttaex.par` for the compiler used.

Interactive Mode

You can use the Targeting Expert interactive mode to achieve different targets to:

- Build an un-configured or optimized target executable by following the steps described in [“Targeting Work Flow” on page 2852](#).
- Configure the distributed Telelogic Tau release and handle the already done settings. Please see below.

Although most of the steps in targeting are supported in the [Targeting Work Flow](#) you sometimes need access to other functionality, for example:

- [Compiler Definition for Compilation](#) to the target library’s known compilers.
- [Communications Link Definition for Compilation](#) (for the inter-communication with the SDL Target Tester’s host application).
- [Handling of Settings](#)
- [Customization](#)

Compiler Definition for Compilation

Note:

All the modifications that can be done here are only valid for the current system, i.e. all the information will be stored into the system’s target directory.

Add

The following is applicable only if using the Cmicro SDL to C compiler:

Select *Add a new Compiler* from the *Edit* menu. The Add compiler dialog is displayed.

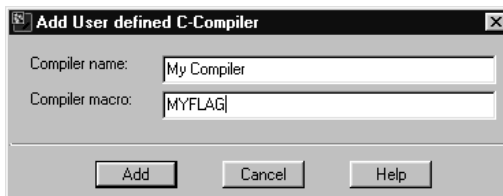


Figure 507: Add compiler dialog

- Compiler Name

The text you enter here will be used to identify the compiler in the Compiler Flag list.

- Compiler Macro

The macro name you enter here will be used to identify the compiler in the target library. The compiler macro must fit to [A-Za-z_] [A-Za-z0-9_]*

When you click *Add* the compiler is added to the `c*_conf.def` file. Please see “Configuration Files” on page 2899 for information about the file’s syntax and duty.

Edit

For each compiler that is supported by a target library there is a specific section in

- `scttypes.h` (Cadvanced), see “Compiler Definition Section in scttypes.h” on page 3069.
- `ml_typ.h` (Cmicro), see “Adaptation to Compilers” on page 3430 in chapter 67, *The Cmicro Library*

Whenever there is a compiler flag defined, not known by the library, a file called `user_cc.h` will be included which has to contain the compiler specific settings.

The dialogs shown below are input masks that request all the needed information to generate such a file. Select *Edit Compiler Section* from the Edit Menu.

Cadvanced

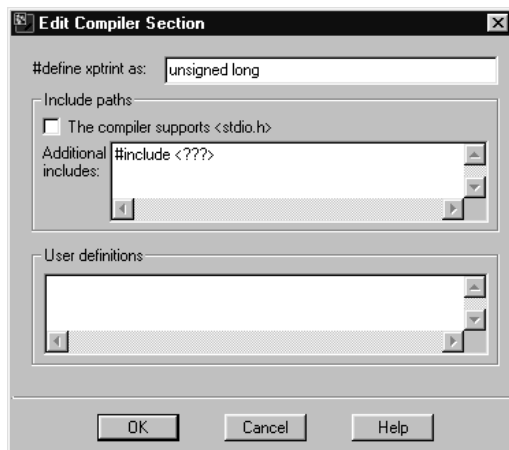


Figure 508: Edit compiler section dialog (Cadvanced)

For Cadvanced the following is requested:

- *#define xprintf*

Enter the mapping for xprintf values; usually `unsigned long`.

- *Include paths*

- *The compiler supports <stdio.h>*

If selected this section will be generated:

```
#ifdef XREADANDWRITEF
#include <stdio.h>
#endif
```

Otherwise nothing will be generated here.

- *Additional includes*

The contents of the edit box will simply be copied in to the `user_cc.h` file. It should carry something like `#include <file>`.

- *User definitions*

Enter user definitions for the `user_cc.h` file.

Cmicro

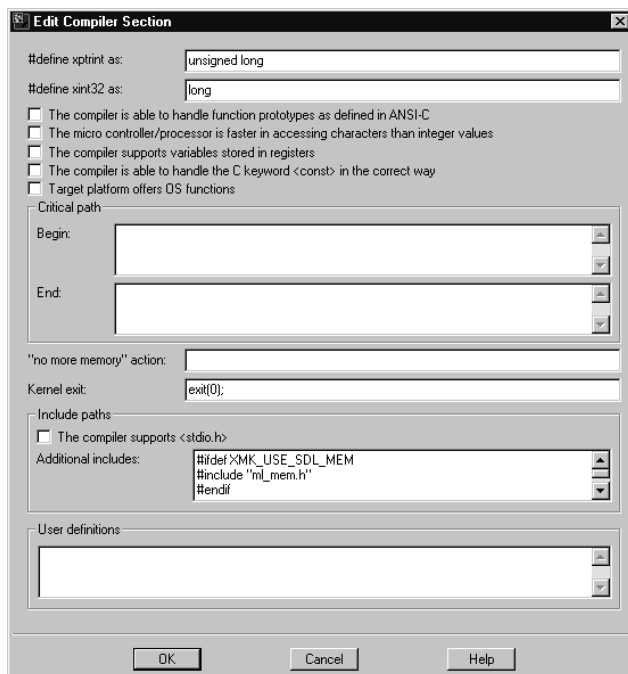


Figure 509: Edit compiler section dialog (Cmicro)

For Cmicro the following is requested:

- *#define xprint*

Enter the mapping for xprint values; usually unsigned long.

- *#define xint32*

Enter the mapping for xint32 values; usually long.

- *The compiler is able to handle function prototypes as in ANSI C*

- If selected this section will be generated:

```
#undef XNOPROTO
#define XPP(x) x
#define PROTO(x) x
```

- If **not** selected this section will be generated:

```
#define XNOPROTO
#define XPP(x)
#define PROTO(x)
```

- *The micro controller/processor is faster in accessing characters than integer values.*
 - If selected this section will be generated:

```
#define xmk_OPT_INT char
```
 - If **not** selected this section will be generated:

```
#define xmk_OPT_INT integer
```
- *The compiler supports variables stored in registers*
 - If selected this section will be generated:

```
#undef X_REGISTER
#define X_REGISTER register
```
 - If **not** selected this section will be generated:

```
#undef X_REGISTER
#define X_REGISTER
```
- *The compiler is able to handle the C keyword <const> in the correct way*
 - If selected this section will be generated:

```
#undef XCONST
#define XCONST const
```
 - If **not** selected this section will be generated:

```
#undef XCONST
#define XCONST
```
- *Target platform offers OS functions*
 - If selected this section will be generated:

```
#define XMK_USE_OS_ENVIRONMENT
```
 - If **not** selected this section will be generated:

```
#undef XMK_USE_OS_ENVIRONMENT
```
- *Critical paths*

Enter the commands which are needed to have critical paths, i.e. how to disable and enable interrupts.

Note:

All the commands you enter will be copied in to `user_cc.h`. If there is more than one line of source code, each line must end with a `\` because the compiler's preprocessor depends on it.

- *“no more memory” action*

Enter the action (e.g. a function call) to process if no more memory can be allocated.

- *Kernel exit*

Enter the function call to process if the kernel should be exited. Default is `exit()`;

- *Include paths*

- *The compiler supports `<stdio.h>`*

If selected this section will be generated:

```
#ifdef XMK_ADD_PRINTF
#include <stdio.h>
#endif
```

Otherwise nothing will be generated here.

- *Additional includes*

The contents of the edit box will simply be copied into the `user_cc.h` file. It should carry something like `#include <file>`.

- *User definitions*

Enter user definitions for the `user_cc.h` file.

Remove

The following is applicable only if using the Cmicro SDL to C compiler:

To remove a compiler from the private `c*_conf.def` file, select *Remove an Unused Compiler* from the Edit Menu

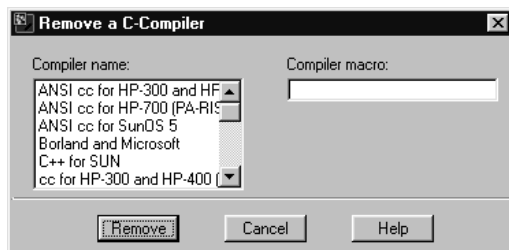


Figure 510: Remove compiler dialog

Note:

The file `user_cc.h` will not be affected by removing a compiler from the `c*_conf.def` file.

Communications Link Definition for Compilation

The following is only applicable if using the Cmicro SDL to C compiler.

Note:

All the modifications that can be done here are only valid for the current system, i.e. all the information will be stored into the system's target directory.

Add

To allow the selection of a new communications link in the Targeting Expert user interface the appropriate macros should be given in the Targeting Expert.

Select *Add a User Defined Communications Link* from the Edit Menu.

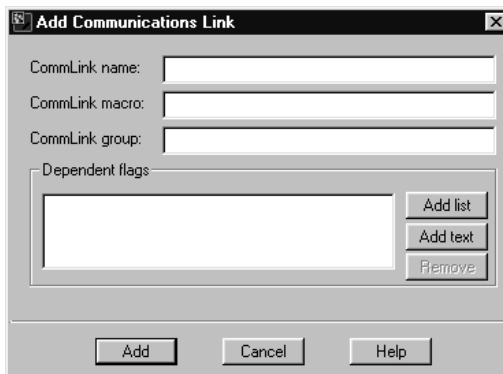


Figure 511: Add a user defined communications link

- *CommLink Name*

The text will be shown in the Targeting Expert UI to identify it in the dialog described in [“Configure the SDL Target Tester \(Cmicro only\)” on page 2873](#).

- *CommLink Macro*

The macro name will be used to identify the communications link in the target library. The communications link macro must fit to [A-Za-z_][A-Za-z0-9_]*

- *CommLink Group*

The text will be used to build a group of the communications link description done here, i.e. the CommLink macro and the dependent flags will be summarized as the group given here.

- *Dependent Flags*

- *Add List*

A dialog where you enter a Value List is displayed.

- *Add Text*

A dialog where you enter a Text Value Flag is displayed.

- *Remove*

The entry you have selected in the list box will be removed.

When you click *Add* the communications link is added to the `c*_conf.def` file. Please see [“Configuration Files” on page 2899](#) for information about the file’s syntax and duty.

Note:

The communications link source code has to be set up by the user.

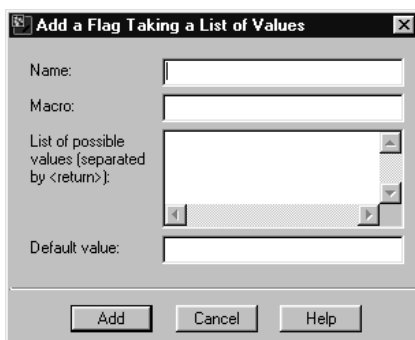
Value List

Figure 512: Add a flag taking a list of values

- *Name*

The name is used for identification purposes in the Targeting Expert user interface.

- *Macro*

The macro will be used for scaling purposes in the target library. It must fit to `[A-Za-z_] [A-Za-z0-9_] *`

- *List of possible values*

Enter a list of all allowed values the macro can take. Each entry has to be delimited by `<return>`.

- *Default value*

Enter the default value which must be one of the possible values entered above.

When you click *Add* this value list is added to the communications link you entered in the dialog in [Figure 511 on page 2843](#).

Text Value Flag

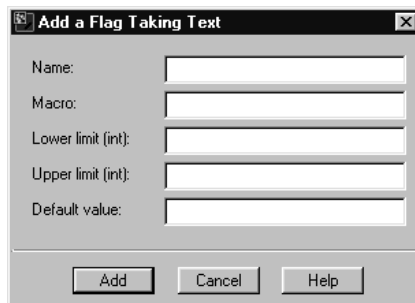


Figure 513: Add a flag taking text

- *Name*

The name is used for identification purposes in the Targeting Expert user interface.

- *Macro*

The macro will be used for scaling purposes in the target library. It must fit to `[A-Za-z_] [A-Za-z0-9_]*`

- *Lower limit*

Enter the lower limit if the value given with this macro is an integer value.

- *Upper limit*

Enter the upper limit if the value given with this macro is an integer value.

- *Default value*

The default value of the macro in this edit line has to be between the lower and upper limit if it is an integer value. Of course, it is possible to enter an alphanumeric value here, too.

When you click *Add* this value list is added to the communications link you entered in the dialog shown in [Figure 511 on page 2843](#).

Remove

To remove a communications link from the private `c_conf.def` file, select *Remove a communications link* from the [Edit Menu](#).



Figure 514: Remove a communications link

Handling of Settings

General

The integration tool bar on the main window offers four buttons to handle the settings, i.e. the way settings are stored and where they are taken from can be influenced.



Figure 515: Integration tool bar in the main window

Note:

The settings addressed in this section are only compiler, linker and make settings. The configuration of the SDL to C compiler library is not touched!

After selecting pre-defined integration settings the Targeting Expert automatically creates a file called `<integration_name>.uis` in the according component directory. (Please see [“Target Sub-Directory Structure” on page 2921](#)).

For some reasons it is sometimes useful to modify this way of saving the user done settings. E.g.



- To be able to “play” with settings it can be useful to switch to another file while the default one is not touched. After playing it is easy to switch back to the old settings by selecting the old file again.



- After doing an adaptation for a specific target the manually modified settings can be exported as new pre-defined integration settings. Please see [“Export” on page 2848](#).



- Estimated there are several components inside of a node and each of them should be build the same way, then it is probably useful to do the adaptation of settings once on node level and to import these settings for each component.

This also means that the compiler, linker and make settings can no longer be modified on component level.



- Estimated there are several components in different nodes that should be build the same way, then it is possible to import the setting from the application.

This also means that the compiler, linker and make settings can not be modified any longer on component level.

Estimated there is an integration 'A' and there are modified settings on this integration on application, node and component level. Then there are three different files `A.uis` saved in the correspondent application, node and component directories.

[Figure 516](#) shows where the settings can be done and where they are taken from when the component is selected.

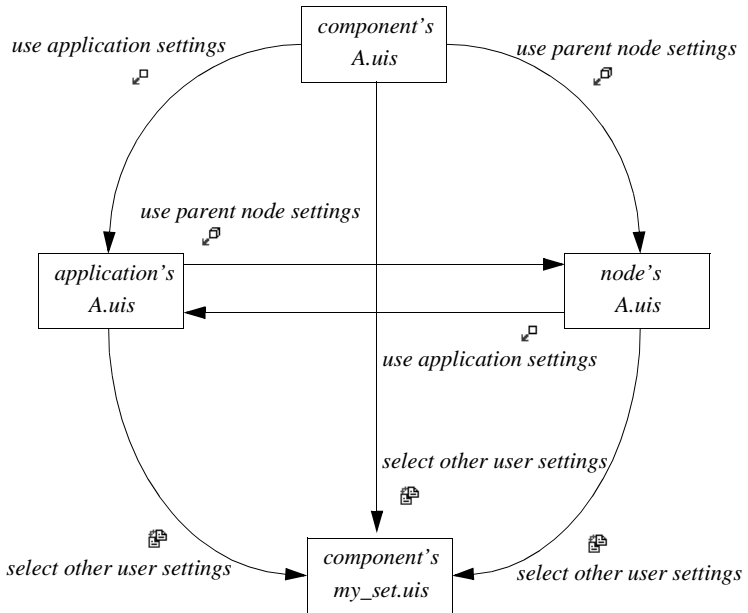


Figure 516: handling the user settings

Export

The settings of the currently selected integration will be exported as new Pre-defined Integration Settings with the name given in the first dialog that pops up.

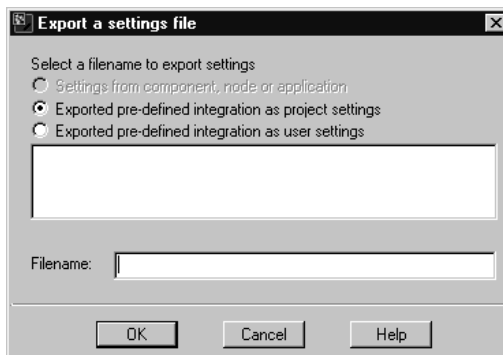


Figure 517

Note:

All the integrations must have unique names, i.e. it is not possible to export settings with an already known name.

- *Export pre-defined integration as project settings*

In this case the new `.its` file is stored in the same directory as the `<systemname>.sdt` file. This has the advantage that all users working on the system can access the pre-defined integration.

- *Export pre-defined integration as user settings*

The `.its` file is stored in the `<installationdir>` in **Windows** or respectively the `$HOME/.telelogic` directory **on UNIX**. This has the advantage that other users do not see all the new pre-defined integration settings.

In the second dialog a list of template files which have to be copied into the target directory can be specified. As default, the list of files from the “parent” settings will be given. Modify the list to your needs. Directly after exporting the settings they can be used for further configurations.

Note:

The sections Configuration Settings to Be Set and Configuration Settings to Be Reset are not exported.

Customization

The Targeting Expert interactive mode can be customized. After selecting the *Customize* entry in the Tools Menu the dialog shown in [Figure 518](#) pops up.

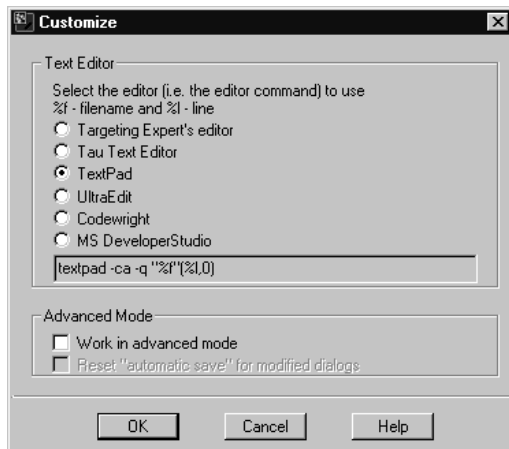


Figure 518: Customize dialog

Text Editor

The text editor which should be opened to edit or view text files can be selected here. The list of supported editors depends on the platform the Targeting Expert is running on (Windows/UNIX). It can be modified or extended in the file `sdttaex.par`. Please see [“Parameter File sdt-taex.par” on page 2925](#).

Advanced Mode

The advanced mode allows you to:

- do more configurations than possible in the non-advanced mode. See [“More Configurations” on page 2851](#) for more information.
- switch between different dialogs without using the *Cancel* or *Save* button
- start the code generation automatically when selecting a new integration

- configure all compilers. Even the ones that are not available on the used platform.

More Configurations

- Normally there is no need to modify the compiler flag if using a pre-defined integration, but in the advanced mode the Compiler Flag can be modified.
- The standard pre-defined integrations do not allow you to use all the configuration options in the Configure and Scale the Target Library dialog (disabled). If the advanced mode is switched on, it is possible to switch on and off all the options supported by the SDL to C compiler library.

Caution!

The pre-defined integrations distributed are tested only with the given library configuration. It is not guaranteed that it will work for all kinds of modifications!

Targeting Work Flow

Introduction

You can start the Targeting Expert form the Organizer's *Generate* menu when having a deployment diagram, the SDL system or a block/process of the SDL system selected.

Estimated the SDL system is selected the Targeting Expert converts the system into a default partitioning diagram model (deployment diagram). This is done because the Targeting Expert can only handle partitioning diagram models as an input. The tree window shown in [Figure 519](#) gives an idea how the default partitioning diagram model looks like.

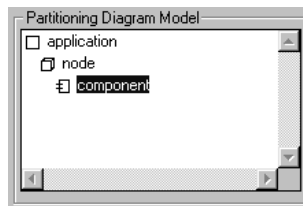





Figure 519: Partitioning diagram in the Targeting Expert

The different entries have got the following meaning:

	application This is the collection of all the deployed SDL systems. It is used as an object to allow configurations of several nodes only, i.e. it is not possible to make an application.
	node A node symbolizes an instance of the platform/computational resource that components execute on, i.e. it is not possible to make a node.
	component A component is interpreted as an executable program.

I.e. an application or node can be configured but not build. Only components can end up in executable programs.

In the start-up phase, directly after the partitioning diagram model has been displayed, the Targeting Expert generates a sub-directory structure into which all the configuration settings, object files and so on will be put. Please see [“Target Sub-Directory Structure” on page 2921](#) for more information.

Now you can use the Targeting Expert to configure each component.

Hint:

If you switch the Targeting Assistant on in the “Help” menu, there are tool tips displayed for each entry on the main window.

Operation Steps

The following operation steps should be done at least once when doing targeting for a component the very first time. When the Targeting Expert is used again later to optimize the target it is of course not necessary to do all the steps once more.

1.

Actions to perform:

- Select the component which should be configured.

Note:

All the settings you define and the actions automatically performed by the Targeting Expert have an influence only on the component selected in the Targeting Expert tree window (see [Figure 519 on page 2852](#)).

If the configuration should be re-used (this is sometimes reasonable if several components should be built using exactly the same settings), you can modify the way your settings are handled. Please see [“Handling of Settings” on page 2846](#).

2.

Actions to perform:

- Select the Pre-defined Integration Settings
- Select an SDL to C Compiler (if not already done in the pre-defined integration settings)
- Select a C Compiler (if not already done in the pre-defined integration settings)

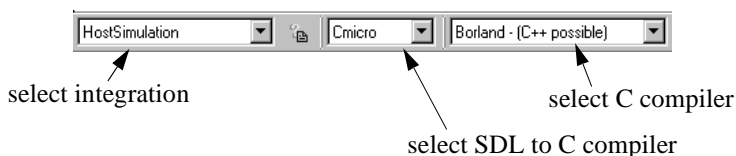


Figure 520: Integration selection in the main window

Select the Pre-defined Integration Settings

Delivered with every Telelogic Tau distribution there are several Pre-defined Integration Settings which you can use to get a target executable in an easy manner.

To get an optimized target executable concerning size and speed these pre-defined integration settings should only be seen as templates.

Please see “Distributed Pre-defined Integration Settings” on page 2904.

The pre-defined integration settings can be selected in the integration tool bar in the main window (see Figure 520). All the pre-defined integration settings are sorted into the groups

- Simulations
- Validations
- Target Tests
- Bare Integrations
- Light Integrations
- Threaded Integrations
- Tight Integrations

Additionally the integration *<user defined>*, which does not belong to an integration type and is not pre-defined, can be selected.

Hint:

If there is no set of pre-defined integration settings that fits your needs (e.g. a not yet supported compiler), it is recommended to select `<user-defined>` and to do the further configurations by hand.

Select an SDL to C Compiler

The SDL to C compiler to be used will automatically be set if you select one of the pre-defined integration settings. It is only necessary to select the SDL to C compiler if the `<user-defined>` settings have been selected. All the available SDL to C compilers will be given in the combo box shown in [Figure 520](#).

Depending on the licenses found the following SDL to C compilers are supported:

- Cadvanced
- Cmicro

Note:

Directly after the SDL to C compiler is selected (i.e. after pre-defined integration settings are selected) the Targeting Expert checks if there is already an automatic configuration file `sdl_cfg.h` (done by the SDL to C compiler) available.

If not, the SDL to C compiler should be invoked to generate an automatic configuration.

For Cadvanced this is only done if the flag `USER_CONFIG` is set in the compiler options (e.g. `-DUSER_CONFIG`).

Select a C Compiler

After a pre-defined integration setting is selected the Targeting Expert tries to automatically set the compiler to be used, i.e.

- if there is only one compiler supported with the selected pre-defined integration settings than that one is set.
- if there are more than one compiler supported the Targeting Expert tries to set the default compiler selected in the Telelogic Tau Preferences. (Please see [chapter 3, The Preference Manager](#))

However, it is possible to set another compiler by selecting it in the combo box shown in [Figure 520](#).

If the *<user defined>* integration has been chosen, a new compiler can be added by selecting the entry “Add new compiler description”.

3.

Actions to perform:

- Configure Compiler, Linker and Make
- Configure and Scale the Target Library
- Configure the SDL Target Tester (Cmicro only)
- Configure the Host (Cmicro only)

To select the desired configuration dialog, please select the correspondent entry in the partitioning diagram.



Figure 521: Tree structure in the partitioning diagram

Configure Compiler, Linker and Make

This part of the configuration is divided into four sub-steps which are taken from the selected pre-defined integration settings as far as possible.

Compiler

You enter the needed compiler configuration in a special input mask. The compiler you specify here will be used to compile the generated code and the target library.

Targeting Work Flow

Compiler Description

Compiler name:

The placeholders %s (source file), %o (object file) and %l (include path) must be used.

Options:

☐ Compile as C++

☐ Compile as debug

Library Flag:

Include:

Comm. Include:

Obj. extension:

C parser and assembler description

C parser name:

The placeholders %s (source file), %o (object file) must be used.

Options:

Input ext.:

Assembler:

The placeholders %s (source file), %o (object file) must be used.

Options:

Input ext.:

Figure 522: Compiler configuration

Click *Default* if you want to restore the default values.

- *Compiler Name*

Enter the name of the compiler application.

- *Options*

Enter the options given to the compiler as command line arguments here.

- Enter the placeholder %s where the source file name of the file to be compiled has to be inserted (used for the makefile generation).
- Enter the dummy parameter %o where the object file's name has to be inserted (used for the makefile generation).
- Enter the dummy parameter %I where the include path option has to be placed (see below).

- *Compile as C++*

The compiler options to compile C files as C++ files will be added/removed from the [Options](#). For a definition of the used compiler options see the "[Parameter File sdttax.par](#)" on page 2925.

- *Compile as debug*

The compiler options to generate the object files including debug information will be added/removed from the [Options](#). For a definition of the used compiler options see the "[Parameter File sdttax.par](#)" on page 2925.

- *Library Flag*

The compiler option to define a flag plus the C macro used to select the desired library must be given here. (When using Cmicro this field is empty per default.)

- *Include*

Enter the compiler option needed to specify include paths and the include paths themselves here. The complete contents of this entry will replace the entry %I in the [Options](#) (see above).

The include path can contain the following environment variables (The environment variables will be expanded during the make execution):

Variable	Path it points to
sdtmdir	<installationdir>/sdt/sdtmdir/<platform> as long as Library directory is not set

Targeting Work Flow

Variable	Path it points to
scttargetdir	Absolute path to the current target directory (depends on the selected component)
sctobjdir	$\$(scttargetdir)$ + the relative path selected in <u>Object directory</u>
sctuseinclude (Cadvanced only)	$\$(sdt\text{dir})/INCLUDE$
sctincludedir (Cmicro only)	$\$(sdt\text{dir})/cmicro/include$
sctkernel\text{dir}	$\$(sdt\text{dir})/cmicro/kernel$
scttester\text{dir}	$\$(sdt\text{dir})/cmicro/tester$

- *Comm Include*

The compiler option needed to specify include paths and the include paths to the coder library must be given here. (Only used if the ASN.1 and/or SDL coder functions are generated and used)

- *Obj. extension*

Enter the object extension to be used for the compiler output.

If only the information up to here is specified in this dialog, each C file is compiled in one single step as shown in [Figure 523](#).

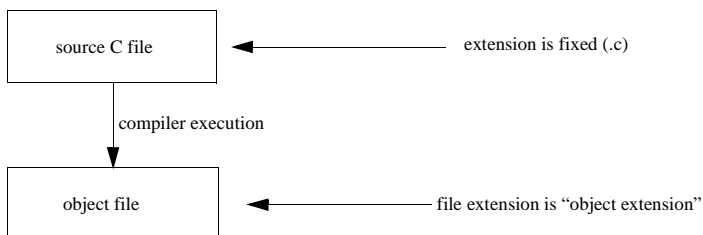


Figure 523:Single step compilation model

For some target compilers, compilation is done in three steps using three different tools. If that is the case, the compilation of each C file is

done like shown in [Figure 524](#). In this case the following entries also need to be given.

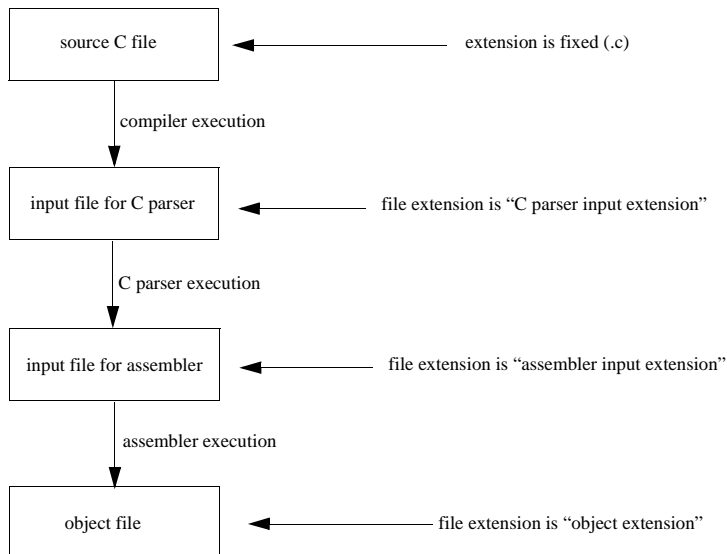


Figure 524: Three step compilation model

- *C parser name*

The name of the C parser application has to be given here.

- *Options (C parser)*

Enter the options given to the C parser as command line arguments here.

- Enter the placeholder `%s` where the input file name of the file to be parsed has to be inserted (used for the makefile generation).
- Enter the dummy parameter `%o` where the output file's name has to be inserted (used for the makefile generation).

- *Input extension (C parser)*

Enter the input file's extension for the C parser.

- *Assembler*

Targeting Work Flow

The third step to process is the execution of the assembler. Enter the name of the assembler application here.

- *Options (Assembler)*

Enter the options given to the assembler as command line arguments here.

- Enter the placeholder `%s` where the input file name of the file to be assemble has to be inserted (used for the makefile generation).
- Enter the dummy parameter `%o` where the output file's name has to be inserted (used for the makefile generation).

- *Input Extension (Assembler)*

The input file's extension used by the assembler has to be specified in this edit line.

Source Files

All the files (except the coder and the generated files) which will be compiled and linked to the target executable are listed here.

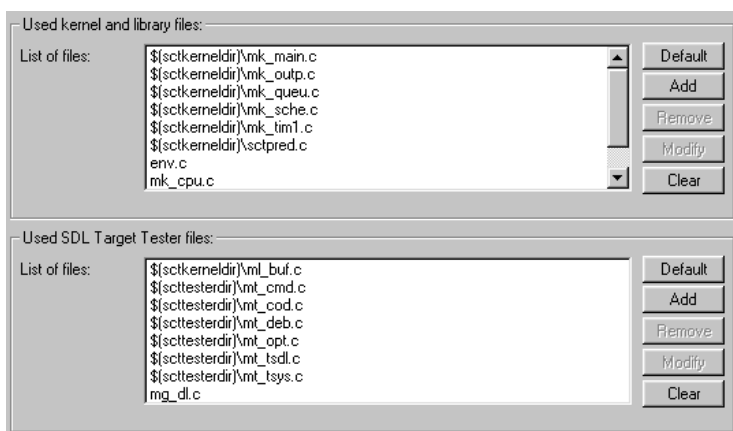


Figure 525: Source files used

- *Used kernel and library files and
Used SDL Target Tester files (Cmicro only)*
 - *Default*

Restores the default values.
 - *Add*

A file selection dialog is opened where you can select the file which should be added to the list of kernel files.
 - *Remove*

Removes the item you have selected in the list box.
 - *Modify*

A dialog pops up and the selected entry can be modified.
 - *Clear*

Deletes all the items in the list box.

Compiler Flag

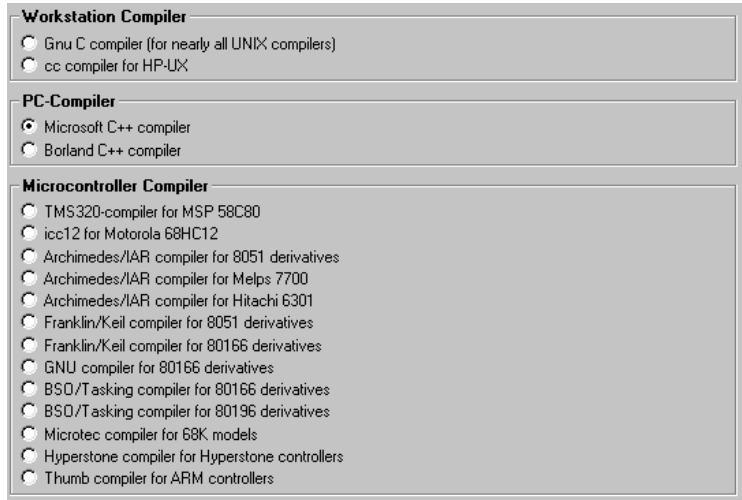
Note:

This section is only available if the Advanced Mode is selected and the SDL to C compiler Cmicro is used.

When compiling the generated code and/or the SDL to C compiler's library you need to set a compiler flag to adjust the code.

The list of available compilers shown in Figure 526 belongs to the Cad-
vanced target library.

Targeting Work Flow



The image shows a screenshot of a software configuration window titled "Targeting Work Flow". It contains three sections for selecting a compiler:

- Workstation Compiler**
 - ☐ Gnu C compiler (for nearly all UNIX compilers)
 - ☐ cc compiler for HP-UX
- PC-Compiler**
 - ☒ Microsoft C++ compiler
 - ☐ Borland C++ compiler
- Microcontroller Compiler**
 - ☐ TMS320-compiler for MSP 58C80
 - ☐ icc12 for Motorola 68HC12
 - ☐ Archimedes/IAR compiler for 8051 derivatives
 - ☐ Archimedes/IAR compiler for Melpis 7700
 - ☐ Archimedes/IAR compiler for Hitachi 6301
 - ☐ Franklin/Keil compiler for 8051 derivatives
 - ☐ Franklin/Keil compiler for 80166 derivatives
 - ☐ GNU compiler for 80166 derivatives
 - ☐ BSO/Tasking compiler for 80166 derivatives
 - ☐ BSO/Tasking compiler for 80196 derivatives
 - ☐ Microtec compiler for 68K models
 - ☐ Hyperstone compiler for Hyperstone controllers
 - ☐ Thumb compiler for ARM controllers

Figure 526: Flag selection

Caution!

You must select the correct compiler flag in order to avoid compilation errors.

For more information concerning the compiler specific adjustment of the libraries see


- `scttypes.h` (Cadvanced), see [“Compiler Definition Section in scttypes.h” on page 3069](#).
- `ml_typ.h` (Cmicro), see [“Adaptation to Compilers” on page 3430 in chapter 67, *The Cmicro Library*](#)

Additional Compiler

If there are other files than the generated ones or the SDL to C compiler's library to be compiled all the requested things in this section have to be entered.



Compiler Description

These settings are to be used for additional files which should not be compiled with the standard compiler.
(Maybe only the compiler options may differ.)

Compiler name:  Default

The placeholders %s (source file), %o (object file) and %I (include path) must be used.





Options: Default

Include:   Default

Obj. extension: Default

Additional files to compile:

List of files: Default

Dependencies: Default

Figure 527: Additional compiler configuration

- *Compiler Name*

Enter the name of the compiler application.

- *Options*

Enter the options given to the compiler as command line arguments here.

- Enter the placeholder %s where the source file name of the file to be compiled has to be inserted (used for the makefile generation).
- Enter the dummy parameter %o where the object file's name has to be inserted (used for the makefile generation).
- Enter the dummy parameter %I where the include path option has to be placed (see below).

- *Include*

Enter the compiler option needed to specify include paths and the include paths themselves here. The complete contents of this entry will replace the entry %I in the Options (see above).

Targeting Work Flow

- *Obj. extension*

Enter the object extension to be used for the compiler output.

- *List of files*

- *Default*

Restores the default values.

- *Add*

A file selection dialog is opened where you can select the file which should be additionally compiled.

- *Remove*

Removes the item you have selected in the list box.

- *Modify*

A dialog is displayed and the selected entry can be modified

- *Clear*

Deletes all the items in the list box.

- *Dependencies*

List the dependencies for the makefile here. Note that the name of the additional file itself is automatically generated as a dependency.

Linker

To link all the compiled files (generated ones, the ones building the library and additional ones) you must configure a linker.

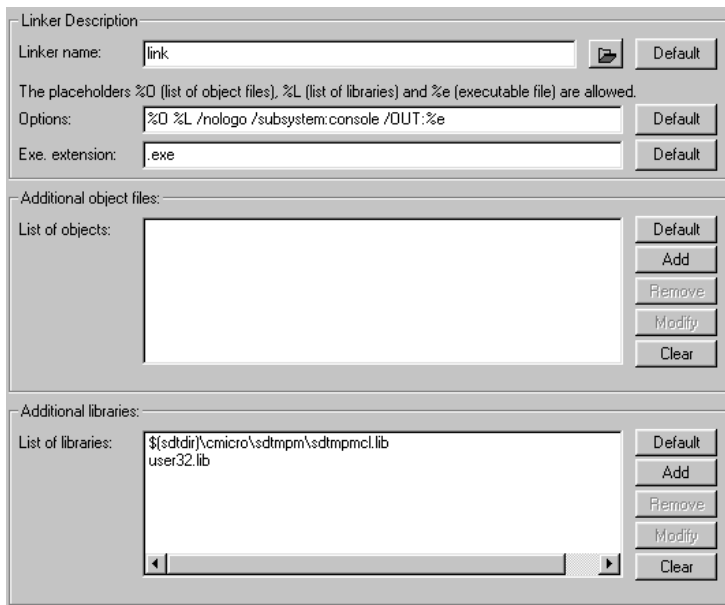


Figure 528: Linker configuration

Click *Default* if you want to restore the default values.

- *Linker name*

Enter the name of the linker application.

- *Options*

Enter the options given to the compiler as command line arguments here.

- Enter the dummy parameter `%O` in the place of the list of all compiled files and the additional object files to link (used for the makefile generation).
- Enter the dummy parameter `%L` in the place of the list of all the libraries (if there are some) which have to be linked.
- Enter the dummy parameter `%e` in the place where the executable file's name has to be inserted (used for the makefile generation).

Targeting Work Flow

- *Exe. extension*

Enter the executable extension to be used for the linker output.

- *Additional object files and Additional libraries*

- *Default*

Restores the default values.

- *Add*

A file selection dialog is opened where you can select the file which should be additionally linked to the executable.

- *Remove*

Removes the item you have selected in the list box.

- *Modify*

A dialog is displayed and the selected item in the list box can be modified.

- *Clear*

Deletes all the items in the list box.

Library Manager

Note:

This section is available only if an SDL and/or ASN.1 coder is selected. Please see [“Communication” on page 2877](#) for details on how to select a coder.

The library manager offers the command that will be used to build a library from all the coder files. The dialog is shown in [Figure 529](#)

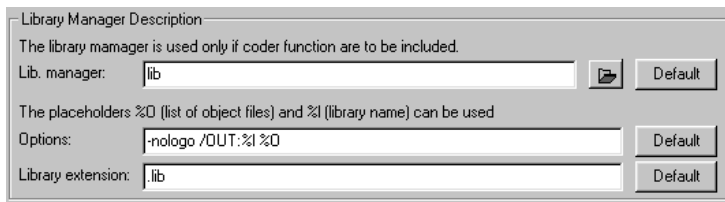


Figure 529: Library Manager Configuration

Note:

If no library manager is given all the coder object files will be linked directly to the target application.

Click *Default* if you want to restore the default values.

- *Lib. Manager*

Enter the name of the library manager application.

- *Options*

Enter the options given to the library manager as command line arguments here.

- Enter the dummy parameter %O in the place of the list of all coder object files to link (used for the makefile generation).
- Enter the dummy parameter %I in the place where the library's name has to be inserted (used for the makefile generation).

- *Library extension*

Enter the library extension here.

Make

You can select the make tool to do all the compile and link actions here.

Make tool

Make tool:

Makefile generation

☒ Generate makefile

Generator:

Makefile:

Makefile generation Parameters

Object directory:

Library directory:

The placeholder %s will be replaced by the component's name,
%c will be replaced by the converter tool file (including path).

Pre-make:

The placeholder %s will be replaced by the source file's name (including path).

Pre-compile:

To use the pre-processor command as pre-compile step press button:

The placeholder %s will be replaced by the executable's name (including path).

Post-link:

Figure 530: Make tool configuration

Click *Default* if you want to restore the default values.

- *Make tool*

Select the make tool you wish to use. Currently the following make tools are supported: (see [“Make Applications” on page 2928](#))

- Borland make (using temporary response file)
- Borland make (ignore exit codes)
- Borland make
- Microsoft nmake (using temporary response file)
- Microsoft nmake (ignore exit codes)
- Microsoft nmake
- UNIX make
- Tasking mk166
- Tornado make

Hint:

“Using temporary response file” means that the compiler’s and the linker’s command line options are passed to the compiler/linker via a temporary file generated by the make tool.

This feature can only be used if the compiler in use supports this facility.

Note:

If the make tool you would like to use is not in the list. Please add a new definition in `sdtttaex.par`. For more information view [“Make Applications” on page 2928](#)

- *Generate makefile*

Select this if a makefile should be generated before make is invoked. The name of the generated makefile is the one in *Makefile*.

- *Generator*

The Targeting Expert is designed to execute external makefile generators whenever the build-in makefile generator (`intern`) is not sufficient.

Enter the name (and the path) of the external makefile generator here. Please see [“External Makefile Generator” on page 2934](#) for further information about how to build an external makefile generator.

- *Makefile*

Enter the name of the makefile.

- *Object directory*

The object directory is the relative path seen from the current target directory, in which the compiler output files (object files) should be written.

Note:

If the directory `<target_directory>/<object_directory>` does not exist it will be automatically created.

- *Library directory*

Targeting Work Flow

The library directory is the absolute path to the target library to be used. Under normal circumstances this field can be left empty as the Targeting Expert automatically uses the path in the Telelogic Tau installation:

```
<installationdir>/sdt/sdtdir/<platform_sdtdir>
```

Note:

The Cmicro target library will be expected in a sub-directory called `cmicro`.

The Cadvanced library will be expected in a sub-directory called `INCLUDE`.

- *Pre-make*

The pre-make step will be executed before any other action specified in the makefile is performed.

- The placeholder `%s` will automatically be replaced by the component's name.

- *Pre-compile*

The pre-compile of the generated file(s) will be executed before they are compiled.

Enter the application which performs the pre-compile here.

- The placeholder `%s` can be used to represent the source files name.
- The preprocessor button inserts the complete command that is necessary to pre-process the generated C file with the default compiler before the compilation with the target compiler.

Hint:

It is probably useful to use the utility functions delivered in combination with the Targeting Expert. Please see [“Utilities” on page 2935](#)

- *Post-link*

The post-link of the linked files will be executed as the last step in the make task.

Enter the application which performs the post-link here. (E.g. a conversion from absolute to Intel-HEX format).

- The placeholder `%s` can be used to represent the linker output file.

Configure and Scale the Target Library

The configuration and the scaling of the used target library is done by setting/resetting compiler macros (`#define` in C). These macros will be called “flags” in the further description.

For more information concerning allowed flags see

- `set_mcf.h` (Cadvanced), see [“Some Configuration Macros” on page 3078](#).
- `ml_mcf.h` (Cmicro), see [“Compilation Flags” on page 3394 in chapter 67, *The Cmicro Library*](#)

The Targeting Expert offers an easy way of setting/resetting these flags. All the allowed flags are divided into different groups, e.g. SDL support and environment.

[Figure 531](#) shows an example for Cmicro.

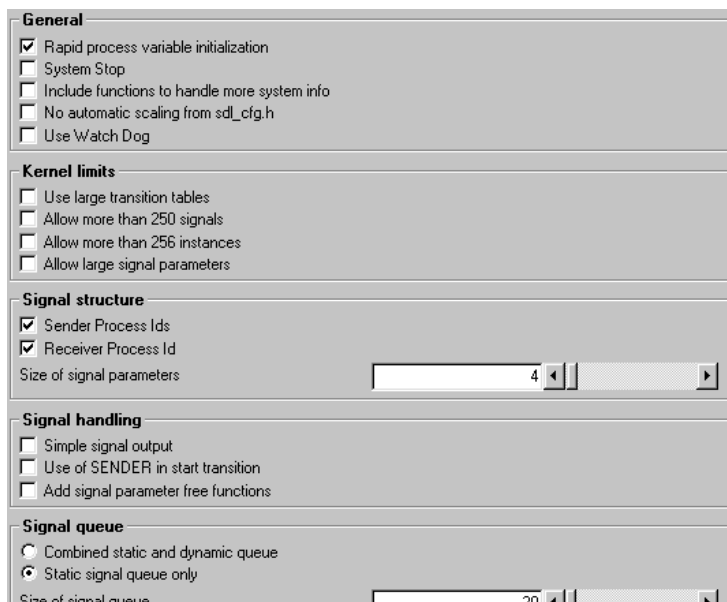


Figure 531: Configuration flag dialog

When all the configurations have been done the Targeting Expert generates

- `ml_mcf.h` (a C header file for Cmicro) which will be included in `ml_typ.h` during the compilation of the corresponding files. This is done for all the pre-defined integrations using the SDL to C compiler Cmicro.
- `sct_mcf.h` (a C header file for Cadvanced) which will be included in `scttypes.h` if the flag `USER_CONFIG` is defined. The flag `USER_CONFIG` is only defined for the pre-defined integrations Application (applclenv) and Application, debug (debclenv-com) using the SDL to C compiler Cadvanced.

Configure the SDL Target Tester (Cmicro only)

The configuration of the SDL Target Tester is very similar to the configuration of the target library.

The settings done for the target will also be generated into the file `ml_mcf.h`. (Please see [“Configure and Scale the Target Library” on page 2872.](#))

Configure the Host (Cmicro only)

The settings for the host will be generated into the file `sdtmt.opt` which will be read by the SDL Target Tester during start-up to get information about the gateway to be used and the target’s memory layout. Please see [“Communication Setup on the Host System” on page 3521 in chapter 68, *The SDL Target Tester*](#) for more information about how to configure the SDL Target Tester’s host application.

4.

Actions to perform:

- [Configure how to Make the Component](#)
- [Make the Component](#)

Configure how to Make the Component

The possible options are divided into four pages:

Note:

All the Analyzer options entered in the Organizer will be re-used by the Targeting Expert.

SDL to C Compiler

SDL to C compiler | Communication | Env. header file | Execution

☒ Analyze/generate code

Analyzer options | Save commands

Code generation options:

The basic options how to generate code have to be set here.

☒ All files

☐ Capitalization lower case

☒ Full variable prefix

☐ Instance information file

Separation: No - do not separate | Default

File name prefix: | Default

Environment:

Note: The environment header file generation cannot be switched off if coder or environment functions are selected.

☐ Environment header file (configuration on other tab)

☐ Environment functions

☐ Signal number file

Filter Analyzer output:

The warning or error message given here will be filtered from the Analyzer output (Example: WARNING 5, ERROR 300)

| Default

Figure 532

Note:

Because the Advanced Mode is switched off there might be some controls disabled because it does not make sense to modify them for the selected integration.

- *Analyze/generate code*

Switch the Analyzer and SDL to C compiler execution on/off when starting the make process.

- *Analyzer options*

The Organizer's Analyzer-Options dialog is displayed and the Analyzer options used for all the integrations can be modified. Please see "Analyze SDL" on page 112 in chapter 2, *The Organizer*.

- *Save commands*

A text file containing all the Analyzer commands configured for the selected integration can be saved. It is possible to start the Analyzer directly using this command file with `sdt san < commandfile`

- *All files*

If selected, the SDL to C compiler will generate all files anew. It does not matter whether the file's contents have changed or not. Unselecting it can economize the build process.

- *Capitalization lower case*

When selected all identifiers are translated to lower case. Otherwise capitalization is used in the declaration of the object

- *Full variable prefix*

All the variables used in SDL will be generated with a full prefix to C to make them unique if this item is selected. This comes with a disadvantage of long variable names.

- *Instance information file*

An instance information file will be generated. Please see "SDL Instance Information" on page 2442 in chapter 55, *The SDL Analyzer*.

- *Separation*

It is possible to select between three alternatives:

- do not separate
- use the user defined separation done in Organizer
- separate each SDL entity into one C file

- *File name prefix*

All the generated files (except those with fixed names like `sdl_cfg.h`) will be generated with the given file name prefix.

- *Environment header file*

The environment header file (`.ifc`) which is used in the coder and environment functions will be generated if switched on.

- *Environment functions*

Environment functions will be generated or not.

Targeting Work Flow

Note:

If the generation of environment functions is switched on it may be needed to add the environment source file to Source Files.

Please see

- “Initializing the Environment / Interface to the Environment” on page 3439 in chapter 67, *The Cmicro Library* for Cmicro
- “Building an Application” on page 2695 for Cadvanced

- *Signal number file* (Cadvanced only)

Generate a signal number file if selected.

- *Conversion style* (Cmicro only)

Select the rules how to generate the converter file here.

- *Filename* (Cmicro only)

Give a name to the converter file

Communication

SDL to C compiler | Communication | Env. header file | Execution

Coders

☐ generate SDL coder functions

☐ generate ASN.1 coder functions

Signal sending

☐ TCP/IP

Figure 533: Communication configuration

- *Generate/do not generate SDL coder functions.*
(Please see “Type description nodes for SDL types” on page 2718 in chapter 58, *Building an Application*.)
- *Generate/do not generate ASN.1 coder functions.*
(Please see “ASN.1 Type Information Generated by ASN.1 Utilities” on page 2767 in chapter 59, *ASN.1 Encoding and De-coding in the SDL Suite*.)
- *TCP/IP signal sending*

Note:

The TCP/IP communication is supported only for a threaded integration using the SDL to C compiler Cadvanced.

The signal sending to other components can be done via a communications link. If it is switched on, a wizard dialog to set up all the needed information pops up (see [Figure 534](#)).

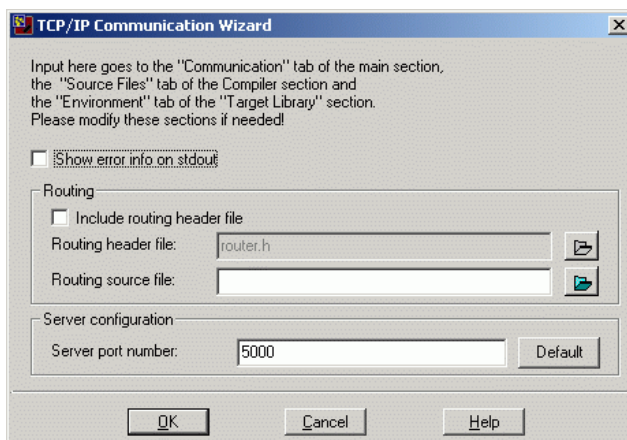


Figure 534: TCP/IP communication wizard

As described in the dialog there are settings done in different sections of the Targeting Expert:

- the manual configuration file `sct_mcf.h`
- the list of source files to compile, i.e. the makefile

Furthermore the generation of

- an [Environment header file](#)
- [Environment functions](#) and
- SDL coder functions

will be switched on.

Caution!

If the TCP/IP communication is switched **on**, a routing source file has to be provided, so that the signals destination can be calculated correctly! Otherwise all the signals will be sent to the sending application.

Caution!

If the TCP/IP communication is switched **off** the routing source file needs to be removed from the list of source files to compile by hand!

Env. Header File

Environment header file generation options

The rules to generate names in the environment header file can be specified here.

The placeholder %n represents the name of the item and cannot be removed.
%s can optionally be added and represents the scope of the item.

☒ Generate section SYNONYMS
%n Default

☒ Generate section LITERALS
%n Default

☒ Generate section TYPEDEFS
%n Default

☒ Generate section OPERATORS

☒ Generate section SIGNALS
%n Default

Note: if modifying this rule, an existing env.c file will be destroyed!

Figure 535: Env. header file configuration

The way the environment header file(. i f c file) is generated can be configured here. Please see “[System Interface Header File](#)” on page 2704 in chapter 58, *Building an Application* for more information.

Note:

The CHANNELS section cannot be configured for Cmicro because the SDL systems structure (blocks and channels) will be lost after code generation.

Execution

Download application

Specify a command line for the download application to use. This download application will automatically be started after the make process is successfully completed.

☒ Use download application

The placeholder %t will be replaced by the target executable's name (including path).

Default

Test application

Select one of the SDL Suite applications or specify the command line of another test application. The test application will automatically be started after the download has been done successfully.

☐ None
☒ SDL Target Tester
☐ Other

The placeholder %t will be replaced by the target executable's name (including path).

Default

Figure 536: Download and test tool configuration

- *Download Application*

Enter also the complete command line which will be necessary to invoke the download application. You can use the placeholder %t which will automatically be replaced by the name of the target executable's name (including the full path).

Note:

The selected download application will automatically be started if make was successful. The Targeting Expert does not try to invoke a download application if “Use download application” is un-checked.

- *Test Application*

There are two kinds of test applications:

- The applications that are part of the SDL suite, i.e. *“The SDL Simulator” on page 2061*, *“The SDL Validator” on page 2229* and *“The SDL Target Tester” on page 3507*.
- The applications from other vendors, e.g. a debugger or emulator.

If you select *Other* you have to enter also the complete command line for the desired test tool. You can use the placeholder %t which

will automatically be replaced by the target executable (including the full path).

Make the Component

The whole make process consists of several different tasks which will automatically be executed one after another. These tasks are:

- [Analyze and Generate Code](#)
- [Generate a Makefile](#)
- [Compile and link](#)
- [Execute](#) (target application or the application in [Download Application](#) and/or [Test Application](#))

The execution of the next task will be stopped if the current task returns with an error. The Targeting Expert event log will give information about the process' state.

Analyze and Generate Code

The selected SDL to C compiler will be invoked and code will be generated for the selected component.

There are two possibilities:

- make
In that case only code for the modified parts of the SDL system will be generated.
- full make
Code for the complete SDL system is generated. Even if there are no modifications.

Generate a Makefile

The makefile to be used will be generated. Please see [“Make” on page 2868](#) for information on how to give the makefile's name and how to switch the makefile generation on or off.

Hint:

If the build-in makefile generation does not fit your needs it is possible to customize the makefile generation.

Please see [“External Makefile Generator” on page 2934](#) to get information on how to build a makefile generator.

Compile and link

The external make tool will be invoked using the generated makefile. Please see [“Make” on page 2868](#) for information on how to select the make tool.

Hint:

To force a compilation of all the C source files the object files have to be deleted first. Please use the *Clean* entry in the [Make Menu](#).

Execute

The target executable will be executed.

Note:

It is sometimes not possible to invoke the target executable from the Targeting Expert, e.g. if the target executable is built for a micro controller. In this case you can select a download application or any other test tool (e.g. debugger) which will be executed instead. Please see [“Download Application” on page 2880](#) and [“Test Application” on page 2880](#).

Hint:

For running test cases the target executable’s output to `stdout` and `stderr` will be re-directed automatically into the Targeting Expert event Log. The target simply should be executed by entering “%t” as the test application (Please see [“Test Application” on page 2880](#)).

Batch Mode

Using the Targeting Expert in the batch mode means starting `taexbatch` instead of `sdttaex`.

The commands have to be given one after another by hand or in a command file. (Please see an [Example of a Batch File](#).) In this case the Targeting Expert has to be started with `taexbatch < commandfile`

Note:

If the commands are given in a command file the Targeting Expert should be started with one of the options `-yes` or `-no` to answer all upcoming questions automatically. Otherwise the application will probably hang up.

Syntax of Batch Mode Commands

Command Names

You may abbreviate a command name by giving sufficient characters to distinguish it from other command names. A hyphen ('-') is used to separate command names into distinct parts.

Abbreviation of Commands

You may abbreviate any part as long as the command does not become ambiguous.

Hint:

Commands used in command files should not be abbreviated because future implementations may conflict with those abbreviations.

Parameters in Commands

Parameters are separated by one or several spaces. Parameters containing spaces have to be given in quotation marks.

Case Insensitivity in Commands

In command names there is no distinction made between upper and lower case letters.

Some more Detailed Description of Parameter Types

- `<flagname>`

Allowed flag names depend on the SDL to C compiler in use.

- For Cadvanced a description of allowed flags can be found in section [“Some Configuration Macros”](#) on page 3078.
- For Cmicro see [“Compilation Flags”](#) on page 3394 in chapter [67, The Cmicro Library](#) for a more detailed description.

`<flagname>` parameters are case sensitive.

- `<boolvalue>`

Allowed boolean values can be `true`, `on` and `yes` which all have the same meaning no matter if written lower case or not. The same applies for the values `false`, `off` and `no`.

- `<stringvalue>`

Any string value is allowed here. There is no check at all if the value can be interpreted correctly. E.g. if “abc” is entered although and integer value is needed it will not be checked but probably cause a compilation error afterwards.

- `<entry>`

Allowed entries depend on the class used in the command. Please see [<class>](#). Parameters of type `<entry>` are case insensitive.

- `<class>`

Allowed classes and the depending entries are shown in the table below. Parameters of type `<class>` are case insensitive.

For all allowed combinations of classes and entries the commands [Get-Setting](#) and [Default-Setting](#) can be used.

- Compiler

Batch Mode

allowed entries	function to set contents
AsCpp	<u>Set-Setting-String</u>
CodInclude	<u>Set-Setting-String</u>
FilesToCompile	<u>Add-Setting-String</u>
Include	<u>Set-Setting-String</u>
LibFlag	<u>Set-Setting-String</u>
Options	<u>Set-Setting-String</u>
Options2	<u>Set-Setting-String</u>
Options3	<u>Set-Setting-String</u>
TesterFilesToCompile	<u>Set-Setting-String</u>
Tool	<u>Set-Setting-String</u>
Tool2	<u>Set-Setting-String</u>
Tool3	<u>Set-Setting-String</u>

– Linker

allowed entries	function to set contents
LibrariesToLink	<u>Add-Setting-String</u>
ObjectsToLink	<u>Add-Setting-String</u>
Options	<u>Set-Setting-String</u>
Tool	<u>Set-Setting-String</u>

– AddCompiler

allowed entries	function to set contents
AddFilesToCompile	<u>Add-Setting-String</u>
Depend	<u>Set-Setting-String</u>
Include	<u>Set-Setting-String</u>
Options	<u>Set-Setting-String</u>
Tool	<u>Set-Setting-String</u>

- Make

allowed entries	function to set contents
Tool	<u>Set-Setting-String</u>

- Global

allowed entries	function to set contents
AddObjectExtension	<u>Set-Setting-String</u>
CodeGenerator	<u>Set-Setting-String</u>
ExecutableExtension	<u>Set-Setting-String</u>
FileNamePrefix	<u>Set-Setting-String</u>
FullSeparation	<u>Set-Setting-Bool</u>
GenerateASNCoder	<u>Set-Setting-Bool</u>
GenerateEnvFunctions	<u>Set-Setting-Bool</u>
GenerateEnvHeader	<u>Set-Setting-Bool</u>
GenerateIfcChannels	<u>Set-Setting-Bool</u>
GenerateIfcLiterals	<u>Set-Setting-Bool</u>
GenerateIfcOperators	<u>Set-Setting-Bool</u>
GenerateIfcSignals	<u>Set-Setting-Bool</u>
GenerateIfcSynonyms	<u>Set-Setting-Bool</u>
GenerateIfcTypedefs	<u>Set-Setting-Bool</u>

Batch Mode

allowed entries	function to set contents
GenerateInstanceInfo	<u>Set-Setting-Bool</u>
GenerateLowerCase	<u>Set-Setting-Bool</u>
GenerateMakefile	<u>Set-Setting-Bool</u>
GenerateSDLCoder	<u>Set-Setting-Bool</u>
GenerateSignalNumber	<u>Set-Setting-Bool</u>
IfcPrefixChannels	<u>Set-Setting-String</u>
IfcPrefixLiterals	<u>Set-Setting-String</u>
IfcPrefixSignals	<u>Set-Setting-String</u>
IfcPrefixSynonyms	<u>Set-Setting-String</u>
IfcPrefixTypes	<u>Set-Setting-String</u>
LibraryDirectory	<u>Set-Setting-String</u>
MakefileGenerator	<u>Set-Setting-String</u>
MakefileName	<u>Set-Setting-String</u>
PreCompile	<u>Set-Setting-String</u>
PreMake	<u>Set-Setting-String</u>
PostLink	<u>Set-Setting-String</u>
ObjectDirectory	<u>Set-Setting-String</u>
ObjectExtension	<u>Set-Setting-String</u>
TestTool	<u>Set-Setting-String</u>

Description of Batch Mode Commands

In this section, the batch mode commands are listed in alphabetical order, along with their parameters and a description.

Add-PR

Parameters:

<pr-filename>

Further PR files can be specified here which should be considered by the SDL to C compiler.

The first PR file has to be given by using Open-PR command.

Add-Setting-String

Parameters:

`<class> <entry> <stringvalue>`

The list <entry> will be extended for the item `<stringvalue>`.

Add-UserSection-String

Parameters:

`<stringvalue>`

The `<stringvalue>` will be added in the user section of `ml_mcf.h` (Cmicro) or respectively `sct_mcf.h` (Cadvanced).

Analyze

Parameters:

`<none>`

The SDL system will be analyzed.

Append-Setting-String

Parameters:

`<class> <entry> <stringvalue>`

`<stringvalue>` will be appended to the current value of `<class> <entry>`.

Clear-UserSection

Parameters:

`<none>`

The complete user section of `ml_mcf.h` (Cmicro) or respectively `sct_mcf.h` (Cadvanced) will be removed.

Default-Setting

Parameters:

`<class> <entry>`

The default value of the given <entry> will be restored.

Exit

Parameters:

`<none>`

The Targeting Expert exits (same as Quit).

Generate-Code

Parameters:

<none>

A code generation for the selected component is done using the SDL to C compiler selected in the associated settings.

Generate-Code-Full

Parameters:

<none>

A full code generation for the selected component is done using the SDL to C compiler selected in the associated settings.

Generate-Makefile

Parameters:

<none>

A makefile will be generated for the selected component. The makefile's name is taken from the associated settings.

Get-Setting

Parameters:

<class> <entry>

The current contents of <entry> will be printed on the screen.

Include

Parameters:

<batch-filename>

All the commands listed in <batch-filename> will be processed.

Help

Parameters:

<none>

A list of all commands with their needed parameters is printed.

Make-All

<none>

For the application or all the nodes and components that are configured (combined or separate configuration) the steps of analysis, code generation, makefile generation and compilation/linkage will be performed.

Make-Clean**Parameters:**

<none>

All the object files will be removed.

Make-Selected**Parameters:**

<none>

The make tool will be executed using the associated make tool and the associated makefile of the selected component.

Open-PDM**Parameters:**

<filename>

The partitioning diagram file <filename> will be loaded.

Open-PR**Parameters:**

<filename>

The PR file <filename> will be loaded and internally converted into a partitioning diagram model.

Open-SDT**Parameters:**

<filename>

The system file <filename> will be loaded and internally converted into a partitioning diagram model.

Prepend-Setting-String**Parameters:**

<class> <entry> <stringvalue>

<stringvalue> will be prepended to the current value of <class> <entry>.

Qualifier**Parameters:**

<SDL-qualifier>

The given <SDL-qualifier> will be used for building a part of the SDL system.

Quit

Parameters:

<none>

The Targeting Expert exits (same as Exit).

Replace-Setting-String

Parameters:

<class> <entry> <old-stringvalue> <new-stringvalue>

The string <old-stringvalue> of <entry> will be replaced with <new-stringvalue>.

Save-Settings

Parameters:

<none>

The modified settings of all the available applications, nodes and components will be saved.

Select

Parameters:

[<application-node-component>]

The entry specified by <application-node-component> in the partitioning diagram will be selected. This step has to be executed before you can generate.

One of the commands Open-PDM, Open-PR or Open-SDT has to be executed before.

Set-Compiler

Parameters:

<compilername>

The given <compilername> will be set as the used compiler. The <compilername> must be available in the default settings of the Set-Integration.

The command Set-Integration has to be executed before.

Set-Flag-Bool

Parameters:

<flagname> <boolvalue>

The SDL to C compiler's library flag <flagname> will be set/reset due to the <boolvalue>.

Caution!

There is no check for interdependencies to other flags.

Set-Flag-String**Parameters:**

`<flagname> <stringvalue>`

The SDL to C compiler's library flag `<flagname>` will be set to the `<value>`.

Caution!

There is no check for upper and lower limits. You must be aware to set a value that is in the allowed range!

Set-Integration**Parameters:**

`<integration>`

The pre-defined integration settings `<integration>` will be set.

- Simulations
 - Performance Simulation
 - Realtime Simulation
 - Simulation
 - TTCN Link
- Validations
 - Validation
- TargetTest
 - Application DEBUG CA
 - HostSimulation
 - HostSimulation (CL)
 - HostSimulation (CLENV)
- Light Integrations
 - Application CA
 - Application CM
 - Application TEST

- Threaded Integrations
 - OSE
 - Solaris
 - VxWorks
 - POSIX/Linux
 - Win32
- Tight Integrations
 - OSE
 - Solaris
 - VxWorks
 - Win32

Note:

To set a Realtime Simulation for example, the `<integration>` has to be given in quotation marks, i.e.

```
Set-Integration "Realtime Simulation"
```

Caution!

In batch mode there is no check if the selected integration fits to the component settings!

Please have a look to [“Distributed Pre-defined Integration Settings” on page 2904](#) for more information.

The command [Select](#) has to be executed before.

Set-Setting-Bool**Parameters:**

`<class> <entry> <boolvalue>`

The boolean [<entry>](#) will be set to its new value.

Set-Setting-String**Parameters:**

`<class> <entry> <stringvalue>`

The non-boolean [<entry>](#) will get the new content `<stringvalue>`.

Start-Download**Parameters:**

`<none>`

The download application selected in Download Application will be invoked.

Start-Testtool

Parameters:

<none>

The test application specified in Test Application will be invoked.

system

Parameters:

<systemcommand>

Executes the <systemcommand> on the underlying OS. The following placeholders can be used in <systemcommand> and will be replaced before execution:

placeholder	... will be replaced by ...
%s	system directory
%b	target directory (the one given in the Organizer)
%t	sub target directory (the one calculated depending on the selected component and integration)
%e	executable name (inclusive extension)
%i	intermediate directory

Write-Log

Parameters:

<filename>

All the output to stdout and stderr will also be written into the file <filename>.

Example of a Batch File

The following batch file can be used to build a Simulator using the Targeting Expert batch mode.

Example 471: Simple batch file

```
Open-SDT access.sdt
select access-node-component
set-integration Simulation
```

Batch Mode

```
set-Compiler Microsoft  
Generate-Code  
Generate-Makefile  
Make-selected  
Exit
```

Note:

The items application, node and component depend on the partitioning diagram model used.

Internal

Partitioning Diagram Model File

The Targeting Expert needs to know how the SDL system which is used for targeting should be partitioned. This is done by using a partitioning diagram model.

Note:

The partitioning diagram model will be generated by the Deployment Editor for deployed systems into a partitioning diagram model file `<partitioning diagram model>.pdm`. It should not be modified by hand! Please see [“Generating Partitioning Diagram Data for the Targeting Expert” on page 1723 in chapter 41, *The Deployment Editor*](#) for more information.

If there is no deployment done for a system, the Targeting Expert generates a default partitioning diagram model for its internal use. This partitioning diagram model is not available as a file. For a default partitioning diagram model there is no chance to specify integration models or threads.

Definitions

The complete SDL system is mirrored in the partitioning diagram’s *application* which takes several *nodes*.

Each *node* can represent a run-time (physical) object with memory and processing capability and is a collection of several *components*.

Each *component* is a collection of different *objects*, with each *object* representing an SDL qualifier. A *component* will result in one executable or OS task.

File Syntax Example

Estimated there is an SDL system like the one shown in [Figure 537](#),

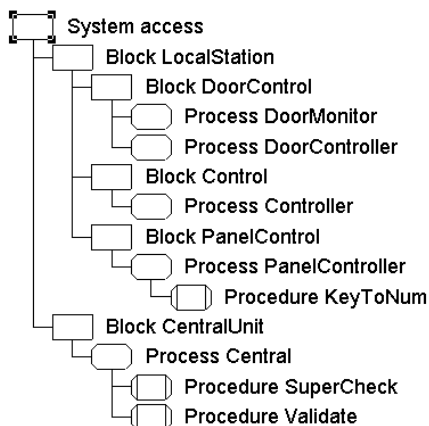


Figure 537: An SDL system

then the simplest possible partitioning diagram model looks like the one shown in [Figure 538](#).

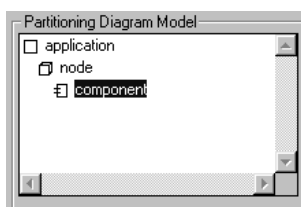


Figure 538: A possible partitioning diagram model

The first few lines of the corresponding partitioning diagram model file is shown in [Example 472](#):

Example 472: Partitioning diagram model file

```
TIMESTAMP: 948787119

APPLICATION*0: access
detail+SystemFileName: access.sdt

NODE*1: node

COMPONENT*2: component
```

```
OBJECT*3: object  
detail+Qualifier: access
```

Explanations

- The `TIMESTAMP` entry is needed for future implementations.
- The `APPLICATION*0: access` gives the name of the used SDL system. The 0 is used as an identifier for internal purposes.
- `detail+SystemFileName` is the name of the used `<system-name>.sdt` file
- `detail+PRFileName` is the name of the used `<systemname>.pr` file. Subsequent entries of `detail+PRFileName` can be used to insert more than one PR file.
- `NODE*1: node` specifies that the following components belong to the node. The node is called `node` and the unique id 1 is used for internal purposes.
- `COMPONENT*2: component` specifies that the following objects belong to this component. The component is called `component` and the unique id 2 is used for internal purposes.
- `OBJECT+5: object` gives one further object which will be assigned to the current component.
- `detail+Qualifier: access` means that all the SDL entities that are sub entities of the qualifier `access` (in this case the complete system) will be part of the `object`.

More Supported Keywords

Details of COMPONENT

- `detail+Integration: Light|Threaded|Tight` specifies in which integration model the component should be build. For backwards compatibility, the keyword `ThreadedLight` is accepted as an alias for `Threaded`.

Details of OBJECT

- `detail+Type: system|block|process` tells which kind of SDL agent the qualifier is all about.

Details of THREAD

- `THREAD: name` is a sub entry of `COMPONENT` and specifies a thread, e.g. for a threaded integration.
- `detail+ThreadPriority: value` specifies the OS priority of the particular thread.
- `detail+StackSize: value` specifies the stack size of the thread. (bytes)
- `detail+QueueSize: value` specifies the maximum number of signals that are allowed in the queue.
- `detail+MaxSignalSize: value` specifies the maximum size of an individual signal in the queue. (bytes)
- `detail+OneThreadPerInstance: true|false` tells if each process instance will become its own thread in a threaded integration if the value is `true`.

Configuration Files

For each SDL to C compiler that will be used there is one configuration file `ca_conf.def` (Cadvanced) and `cm_conf.def` (Cmicro).

The configuration files for all SDL to C compilers can be found in `<installationdir>/sdt/sdtdir`.

The `c*_conf.def` files are ASCII text files that can be (should be) extended by you to adapt a new compiler or to adapt a new communications link (Cmicro only).

You can find all the known configuration flags in:

- [“Some Configuration Macros” on page 3078](#) (Cadvanced)
- [“Compilation Flags” on page 3394 in chapter 67, *The Cmicro Library*](#) (for Cmicro)

Structure of the Configuration File `cm_conf.def`

Note:

All the descriptions in this section are valid for the SDL to C compiler Cmicro only.

Compiler

All the compilers currently supported by a SDL to C compiler library are listed in the `c*_conf.def` file like displayed in [Example 473](#).

Example 473: Structure of a compiler entry

```
BEGIN
  IAR_C51
  IAR_systems compiler of 8051
  COMPILER
  Microcontroller compiler
END
```

The first and the last line of the example simply marks the start and the end of one entry.

Line 2 is the flag which will be used to include the appropriate compiler section in the used library (see `ml_typ.h` for Cmicro and `scctypes.h` for Cadvanced).

Line 3 is the text which will be shown in the Targeting Experts user interface.

Line 4 must be `COMPILER` in this case.

Line 5 is the compiler group this compiler should be associated with. In the distributed files `c*_conf.def` there are three groups, namely:

- Micro controller compiler
- PC compiler
- Workstation compiler

Communications Link

Note:

This section is valid only if the Cmicro SDL to C Compiler is used.

The example below shows the structure of one entry.

Example 474: Structure of a `cm_conf.def` entries

```
1 BEGIN
2   XMK_USE_V24
3   V24 communication
4   COMMLINK
5   V24 interface
6   AUTOSET
7     XMK_V24_BAUD
8     XMK_V24_DEVICE
9   END
10 END
```

Lines 1 and 10 mark the start and the end of one entry.

Line 2 gives the name of the flag as it appears in the generated file `ml_mcf.h`. According to this example line 3 is the marker which appears in the Targeting Expert graphical user interface.

Line 4 specifies the main group this entry belongs to. It has to be `COMMLINK` in this case.

Line 5 gives the subgroup of the entry. This subgroup can be anything you like. In the delivered version there is only the subgroup `V24 interface`. However, you can define your own communications link `CAN Bus` for example.

The lines 6 and 9 belong together and between these lines all flags are listed which have to be set when the flag on line 2 is set. The amount of flags (lines 7 and 8 in the example) is free. In the same way as `AUTOSET` in line 6 there can also be the entries `RESET` or `DEPEND`.

`RESET` means the following flags will be reset if this flag is defined and `DEPEND` means this flag can only be set if the following flags (lines 7 and 8) are already defined.

Not given in the example above is the section `VALUE`, (like `AUTOSET` finished by an `END`) which assign a value to the flag. This section can include three lines of information. The first gives the default value. And, if the value is numeric, the second line gives the lowest and the third one the highest value. You are asked to have a look into `c*_conf.def`.

Also not given in the example above is the section `VALUE_LIST` (finished by an `END`) which assign a value to the flag with a list of allowed values given. The first value is the default value, the following values

(with no upper limit) are giving the list of allowed values. The default value must be given in the list again.

Pre-defined Integration Settings

Introduction

Pre-defined integration settings must be seen as a set of default values which can be used for targeting purposes.

There are several pre-defined integration settings distributed with each Telelogic Tau installation.

- Each set of predefined integration settings is stored in an ASCII file with the extension `.its`.
- The amount of pre-defined integration settings that are to be handled by the Targeting Expert can simply be enlarged by copying a new file `<name>.its` or by exporting own settings. (See [“Export” on page 2848](#))
- The integration names have to be unique!
- The Targeting Expert searches for `.its` files in the following directories:

procedure Search_order_on_UNIX

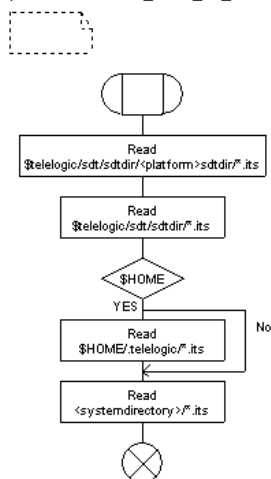


Figure 539: Search order on UNIX

procedure Search_order_in_Windows

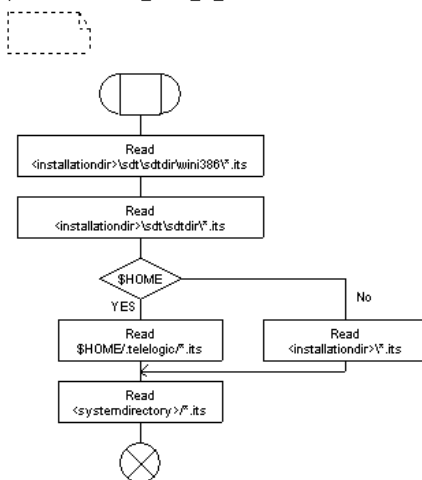


Figure 540: Search order in Windows

Distributed Pre-defined Integration Settings**For Use with Cadvanced SDL to C Compiler**

- Simulation (`debcom`)

Supported compiler	On Sun: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc – Sun Workshop CC On HP: <ul style="list-style-type: none"> – HP GNU gcc – HP cc – HP aCC In Windows: <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-
Timers	Timers are not implemented, i.e. timers set in the SDL system will expire at once
Environment	Will be handled in the Simulator UI
More info	See chapter 50, The SDL Simulator .

- Realtime Simulation (`debcl.com`)

Supported compiler	On Sun: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc – Sun Workshop CC On HP: <ul style="list-style-type: none"> – HP GNU gcc – HP cc – HP aCC In Windows: <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-

Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Will be handled in the Simulator UI
More info	See chapter 50, <i>The SDL Simulator</i> .

- Performance Simulation (perfsim)

Supported compiler	On Sun: <ul style="list-style-type: none">– Sun GNU gcc– Sun Workshop cc– Sun Workshop CC On HP: <ul style="list-style-type: none">– HP GNU gcc– HP cc– HP aCC In Windows: <ul style="list-style-type: none">– Borland C++– Microsoft VC++
Used kernel directory	-
Timers	Not handled
Environment	Not handled
More info	See chapter 64, <i>The Performance Library</i> .

- TTCN Link

Supported compiler	On Sun: <ul style="list-style-type: none">– Sun GNU gcc– Sun Workshop cc On HP: <ul style="list-style-type: none">– HP GNU gcc– cc In Windows: <ul style="list-style-type: none">– Borland C++– Microsoft VC++
Used kernel directory	<code>\$(sdt_dir)/SCT*TTCNLINK</code>

More info	See <u>chapter 36, <i>TTCN Test Suite Generation</i></u> .
-----------	--

- Validation

Supported compiler	On Sun: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc On HP: <ul style="list-style-type: none"> – HP GNU gcc – HP cc In Windows: <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	<code>\$(sdttdir)/SCT*VALIDATOR</code>
More info	See <u>chapter 53, <i>The SDL Validator</i></u> .

- Application (applclenv)

Supported compiler	On Sun: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc – Sun Workshop CC On HP: <ul style="list-style-type: none"> – HP GNU gcc – HP cc – HP aCC In Windows: <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Has to be implemented by the user.
More info	See <u>“Compilation Switches” on page 3041 in chapter 62, <i>The Master Library</i></u> .

- Application, debug (debclenvcom)

Supported compiler	On Sun: <ul style="list-style-type: none">– Sun GNU gcc– Sun Workshop cc– Sun Workshop CC On HP: <ul style="list-style-type: none">– HP GNU gcc– HP cc– HP aCC In Windows: <ul style="list-style-type: none">– Borland C++– Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Has to be implemented by the user.
More info	See <u>“Compilation Switches” on page 3041 in chapter 62, <i>The Master Library</i>.</u>

- Threaded integrations for the platforms
 - OSE (Softkernel Solaris and Windows)
 - Solaris
 - VxWorks (Softkernel Solaris and Windows)

Note: (VxWorks only)

The environment variable `WIND_BASE` has to be set on your system!

- Win32
- OSE (Softkernel Solaris)

Note: (OSE only)

The environment variable `OSE_ROOT` has to be set on your system!

Supported compiler	OSE: <ul style="list-style-type: none"> – Sun GNU gcc – Microsoft VC++ Solaris: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc – Sun Workshop CC VxWorks: <ul style="list-style-type: none"> – Tornado ccsimpc – Tornado ccsimso Win32: <ul style="list-style-type: none"> – Microsoft VC++
Used kernel directory	-
Timers	OS timer
More info	See “Threaded Integration” on page 3225 in chapter 65, <i>Integration with Operating Systems</i> .

- Tight integrations for the platform
 - OSE (Softkernel Win32)

Note: (OSE only)

The environment variable `OSE_ROOT` has to be set on your system!

- Solaris
- VxWorks (Softkernel Win32)

Note: (VxWorks only)

The environment variable `WIND_BASE` has to be set on your system!

- Win32

Supported compiler	OSE: <ul style="list-style-type: none"> – Microsoft VC++ Solaris: <ul style="list-style-type: none"> – Sun GNU gcc VxWorks: <ul style="list-style-type: none"> – Tornado ccsimpc Win32: <ul style="list-style-type: none"> – Microsoft VC++
--------------------	---

Used kernel directory	OSE: sdt/sdtdir/RTOS/OSDelta Solaris: sdt/sdtdir/RTOS/Solaris VxWorks: sdt/sdtdir/RTOS/VxWorks Win32: sdt/sdtdir/RTOS/Win32
Timers	OS timer
More info	See “Tight Integration” on page 3249 in <u>chapter 65, Integration with Operating Systems</u> .

For Use with the Cmicro SDL to C Compiler

- Application (applc1env)

Supported compiler	On Sun: <ul style="list-style-type: none">– Sun GNU gcc– Sun Workshop cc– Sun Workshop CC On HP: <ul style="list-style-type: none">– HP GNU gcc– HP cc– HP aCC In Windows: <ul style="list-style-type: none">– Borland C++– Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Has to be implemented by the user.
Template files	Taken from \$(sdtdir)\cmicro\template
More info	The target executable can be executed on a command line.

- Application including Target Tester and communication via sockets

Supported compiler	<p>On Sun:</p> <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop cc – Sun Workshop CC <p>On HP:</p> <ul style="list-style-type: none"> – HP GNU gcc – HP cc – HP aCC <p>In Windows:</p> <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Has to be implemented by the user.
Template files	Taken from \$(sdttdir)\cmicro\template
More info	The target executable has to be started manually after the Target Tester is started by the Targeting Expert.

- Host Simulation (debcom)

Supported compiler	<p>On Sun:</p> <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop CC <p>On HP:</p> <ul style="list-style-type: none"> – HP GNU gcc – HP aCC <p>In Windows:</p> <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-
Timers	Timers are not implemented, i.e a timer set in the SDL system will never expire.

Environment	Will be handled in the SDL Target Tester UI.
Template files	Taken from \$(sdttdir)\cmicro\template
More info	The target is executed as the SDL Target Tester's gateway. After a complete make <u>The SDL Target Tester</u> will be started and the target can be tested.

- Real-time Host Simulation (debclcom)

Supported compiler	On Sun: <ul style="list-style-type: none">– Sun GNU gcc– Sun Workshop CC On HP: <ul style="list-style-type: none">– HP GNU gcc– HP aCC In Windows: <ul style="list-style-type: none">– Borland C++– Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Will be handled in the SDL Target Tester UI.
Template files	Taken from \$(sdttdir)\cmicro\template
More help	The target is executed as the SDL Target Tester's gateway. After a complete make <u>The SDL Target Tester</u> will be started and the target can be tested.

- Real-time Host Simulation with environment (debclenvcom)

Supported compiler	On Sun: <ul style="list-style-type: none"> – Sun GNU gcc – Sun Workshop CC On HP: <ul style="list-style-type: none"> – HP GNU gcc – HP aCC In Windows: <ul style="list-style-type: none"> – Borland C++ – Microsoft VC++
Used kernel directory	-
Timers	Timers are implemented in a way that one timer tick is equal to one second, i.e. a timer set to 10 in the SDL system will expire after 10 seconds.
Environment	Has to be implemented by the user.
Template files	Taken from \$(sdtldir)\cmicro\template
More help	The target is executed as the SDL Target Tester's gateway. After a complete make <u>The SDL Target Tester</u> will be started and the target can be tested.

Syntax of the .its Files

There is a more or less complex syntax to be followed in .its files which will be described here. In principle each .its file is divided into four sections:

- The Description about the Settings
- The Settings which are once more split into
 - the Global Settings (A)
 - the Compiler specific Definitions
- The Configuration Settings to Be Set
- The Configuration Settings to Be Reset

You can find further syntax information in "Comments and empty Lines" on page 2920 and "Other Rules" on page 2921.

Description about the Settings

The Targeting Expert will read in the description and display it in the user interface whenever a set of pre-defined integration settings is selected.

This section starts with [DESCRIPTION] and has got the only entry `HelpLink`. The description is a link name to this manual.

The Settings

The section of the `.its` files which contains the settings starts with [SETTINGS]. They must be ordered with the [Global Settings \(A\)](#) first, followed by the [Compiler specific Definitions](#).

- Global Settings (A)

The global settings start with the `<keyword1>` equal to `Preferences` and the delimiter '+', i.e. an allowed line in this section looks like:

```
Preferences+<keyword2>: parameter
```

Example 475: Pre-defined integration settings (2A)

```
Preference+IntegrationSettings: <name>
```

The meaning of `<keyword2>` can be found in the table below.

Keyword 2	Meaning of the parameter
IntegrationSettings	The name of this pre-defined integration setting is entered here.

- Global Settings (B)

The global settings part B start with the `<keyword1>` equal to the name of the pre-defined integration settings (see [Global Settings \(A\)](#)) and the delimiter '+', i.e. an allowed line in this section looks like:

```
<name>+<keyword2>: parameter
```

Example 476: Pre-defined integration settings (2B)

```
Simulation+CompilersUsed:      Microsoft
Simulation+CodeGenerator:      Cadvanced
Simulation+GenerateMakefile:   YES
```

The meaning of <keyword2> can be found in the table below.

Keyword 2	Meaning of the parameter
CompilersUsed	The name of a compiler which is used to build the target application must be entered here. This line can appear more than once per file to allow the description of more than one compiler.
CodeGenerator	The name of the SDL to C compiler this pre-defined integration settings are designed for must be entered to the right of this keyword. Currently the SDL to C compilers <ul style="list-style-type: none"> - Cadvanced - Cmicro are possible here.
GenerateAllFiles	All the files will be re-generated even if the contents have not changed (YES) or not (NO)
GenerateMakefile	The makefile generation should be switched on (YES) but can of course be switched off by specifying NO here.
GenerateLowerCase	The SDL to C compiler should generate all SDL identifiers in lower case (YES) or as defined (NO)
GenerateEnvFunctions	The SDL to C compiler should generate template environment functions (YES) or not (NO)
GenerateEnvHeader	The SDL to C compiler should generate an environment header file (YES) or not (NO)
GeneratePrefix	All variables will be generated with full prefix. Default is (YES) but can of course be switched off by specifying NO here.
GenerateSignalNumber	The SDL to C compiler should generate a signal number file (YES) or not (NO)
GenerateSDLCoder	The SDL to C compiler should generate an SDL coder functions file (NO, ASCII or <user>)
GenerateASNCoder	The SDL to C compiler should generate an ASN.1 coder functions file (NO, BER, PER or <user>)

Keyword 2	Meaning of the parameter
Separation	The SDL to C compiler shall generate the C files separated according to the given value: NO, USER, FULL
Analyze	The Analyzer/SDL to C compiler should be invoked during the make (YES) or not (NO)
TestTool	The Test application to be used can be entered here, i.e. the entries Simulator, Validator or Tester but also a complete command line.
FileNamePrefix	The SDL to C compiler will generate all files with the given filename prefix.
GenerateIfcSynonyms	The SDL to C compiler should generate the SYNONYMS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)
GenerateIfcLiterals	The SDL to C compiler should generate the LITERALS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)
GenerateIfcTypedefs	The SDL to C compiler should generate the TYPEDEFS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)
GenerateIfcOperators	The SDL to C compiler should generate the OPERATORS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)
GenerateIfcSignals	The SDL to C compiler should generate the SIGNALS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)
GenerateIfcChannels	The SDL to C compiler should generate the CHANNELS section into the env. header file (YES) or not (NO). (depends on GenerateEnvHeader)

Keyword 2	Meaning of the parameter
IfcPrefixSynonyms	The SDL to C compiler will generate SYNONYMS using the specified rule (Default is %n)
IfcPrefixLiterals	The SDL to C compiler will generate SYNONYMS using the specified rule (Default is %n)
IfcPrefixTypes	The SDL to C compiler will generate SYNONYMS using the specified rule (Default is %n)
IfcPrefixSignals	The SDL to C compiler will generate SYNONYMS using the specified rule (Default is %n)
IfcPrefixChannels	The SDL to C compiler will generate SYNONYMS using the specified rule (Default is %n)
SuppressText	The Analyzer error or warning message to suppress can be given here.

- Compiler specific Definitions

This segment has got all the definitions for an unlimited amount of compilers. All the keywords start with

<name of the setting>+<compiler-name>*

followed by one or two further keywords. If it is followed by two keywords both are separated by '~'.

Example 477: Pre-defined integration settings (3)

```
Simulation+Microsoft*Compiler-Tool:      cl
Simulation+Microsoft*Compiler-Options:  -nologo %I -ML -c -D_Windows -DSCTDEBCOM
/Fo%o %s
Simulation+Microsoft*Compiler-Include:  -I$(sctuseinclude)
Simulation+Microsoft*Compiler-Flag:     IC86
Simulation+Microsoft*Linker-Tool:       link
Simulation+Microsoft*Linker-Options:    -nologo -subsystem:console %O /OUT:%e
Simulation+Microsoft*ObjectExtension:   _smc.obj
Simulation+Microsoft*ExecutableExtension: _smc.exe
Simulation+Microsoft*FileToCompile:     $(sctdir)\INCLUDE\sctsd1.c
Simulation+Microsoft*FileToCompile:     $(sctdir)\INCLUDE\sctpred.c
Simulation+Microsoft*FileToCompile:     $(sctdir)\INCLUDE\sctos.c
Simulation+Microsoft*FileToCompile:     $(sctdir)\INCLUDE\sctmon.c
Simulation+Microsoft*FileToCompile:     $(sctdir)\INCLUDE\sctutil.c
Simulation+Microsoft*FileToCompile:     $(sctdir)\SCTADEBCOM\sctpost.c
Simulation+Microsoft*LibrariesToLink:   $(sctdir)\INCLUDE\msvc50\libpost.lib
Simulation+Microsoft*LibrariesToLink:   user32.lib
Simulation+Microsoft*Make-Tool: Microsoft nmake (using temporary response file)
Simulation+Microsoft*MakefileName:      my_makefile.m
Simulation+Microsoft*MakefileGenerator: -
```

Internal

Simulation+Microsoft*ObjectDirectory: objects

All the keywords and the allowed combinations can be found in the table below.

Keyword 3	Keyword 4	Meaning of the parameter
Compiler	Tool	Name of the compiler application
Compiler	Tool2	Name of the C parser application (optional)
Compiler	Tool3	Name of the assembler application (optional)
Compiler	Options	Compiler command line options (Please see “Compiler” on page 2900)
Compiler	Options2	C parser command line options (optional)
Compiler	Options3	Assembler command line options (optional)
Compiler	LibFlag	The compiler’s option to set defines plus the library flag (e.g. SCTDEBCOM)
Compiler	Include	The compiler’s option for include paths and the include paths
Compiler	CodInclude	The compiler’s option for include paths and the include paths to the coder files
Compiler	Flag	Compiler flag (Please see “Compiler Flag” on page 2862)
Linker	Tool	Name of the linker application
Linker	Options	Linker command line options (Please see “Linker” on page 2865)
ObjectExtension		Extension of the object files
ExecutableExtension		Extension of the target executable
FilesToCompile		Kernel, library and template files to be compiled. This entry can be used more than once.

Keyword 3	Keyword 4	Meaning of the parameter
TesterFilesToCompile		Tester files to be compiled. This entry can be used more than once. (Cmicro only)
FilesToCopy		Template files which should be copied into the target directory. Environment variable <code>\$(sdt_dir)</code> can be used
ObjectsToLink		Additional object files to be linked. This entry can be used more than once.
LibrariesToLink		Additional libraries to be linked (e.g. the compiler's libraries). This entry can be used more than once.
Make	Tool	<p>The name of the make application. This can be:</p> <ul style="list-style-type: none"> • Borland make (using temporary response file) • Borland make • Microsoft nmake (using temporary response file) • Microsoft nmake • UNIX make
MakefileName		The name of the makefile to be generated (or respectively to be used) is entered here.
MakefileGenerator		The Targeting Experts supports the use of external makefile generators which is to be entered here. An <code>intern</code> means that the build-in makefile generator is to be used
ObjectDirectory		The relative path (seen from the target directory) for the object directory can be entered here. If not specified Targeting expert will generate the makefile in a way that the object files will be written into the target directory when make is executed.

Keyword 3	Keyword 4	Meaning of the parameter
LibraryDirectory		If this entry is specified the SDL to C compiler's library will be taken from the specified path. If not specified it will be taken from <code>\$sdt_dir(%SDTDIR%)</code> .
Download	Tool	The full command line to invoke the download application.
PreMake		Action to perform before the target application will be made in the makefile.
PreCompile		If the generated code should be modified before it is compiled, an appropriate application (complete command line) should be entered here.
PostLink		If the linked target application needs to be modified (e.g. conversion into HEX format), the appropriate application (complete command line) should be entered here.
AddCompiler	Tool	For the additional files (see <code>FilesToCompile</code>) the name of the compiler application must be entered here.
AddCompiler	Options	The command line options for the compiler in <code>AddCompiler</code> must be specified here.
AddCompiler	Include	The compiler's option for include paths and the include paths for the compiler entered in <code>AddCompiler</code>
AddCompiler	Depend	The dependencies for the additional files (used inside of the generated makefile).
AddObjectExtension		The object extension of the <code>AddCompiler</code> must be entered here.
AddFilesToCompile		Kernel files to be compiled (e.g. communications link). This entry can be used more than once.

Default Configuration Settings

For several pre-defined integrations it is necessary to have some configuration flags defined as default. In this section a simple list of all configuration flags which should be set is given.

This section starts with `[CONFIGURATION_DEFAULT]`.

Please see *[“Configuration Files” on page 2899](#)* for more information about configuration flags.

Configuration Settings to Be Set

For several pre-defined integrations it is necessary to have some configuration flags set without giving you the chance to un-select them. In this section a simple list of all configuration flags which have to be set is given.

This section starts with `[CONFIGURATION_SET]`.

Please see *[“Configuration Files” on page 2899](#)* for more information about configuration flags.

Configuration Settings to Be Reset

In opposite to the configuration flags which have to be set, it is of course necessary sometimes to prevent the selection of other flags. This can be done in this section by simply listing all the flags that have to be reset.

This section starts with `[CONFIGURATION_RESET]`.

Please see *[“Configuration Files” on page 2899](#)* for more information about configuration flags.

SDL to C Compiler Settings to Be Disabled

For some pre-defined integrations it does not make sense to configure all the SDL to C compiler options available. It is possible here to list all the options to be disabled in the user interface.

This section starts with `[SDL_TO_C_COMPILER_DISABLED]`.

Comments and empty Lines

- Empty lines are allowed everywhere in the `an .its` file.

- Comments are allowed everywhere in the `.its` file but have to start with an `'#'` in the first column.

Other Rules

- Only one set is allowed per `.its` file
- The amount of compilers supported for each set is unlimited.
- The characters `'['`, `']'`, `'+'`, `'*'`, `'~'` and `':'` are forbidden in compiler names or the name of the pre-defined integration settings itself.

User-defined Integration Settings

All the settings done by the user will be stored in separate files into the target directory, i.e.:

- The complete set of settings will be stored (not the difference to the defaults)
- There is one file for each integration configuration done for an application, node or component
- If several nodes or components have to use the same settings it is possible to manually select one file for them.
Please see [“Handling of Settings” on page 2846](#).

Target Sub-Directory Structure

The Targeting Expert automatically sets up a sub-directory structure for all the applications, nodes and components. Furthermore there is a sub directory for each kind of integration done.

Example 478: Sub-directory structure

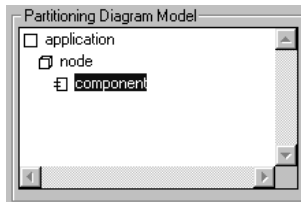


Figure 541: Example of a partitioning diagram

Estimated there is a partitioning diagram which looks like [Figure 541](#), the Targeting Expert generates a sub-directory structure like:

```
<target_dir>
+ application._0
+ node._1
+ component._2
```

Note:

The suffixes `._*` give unique identifiers which make it possible to rename the application, nodes or components and to keep the done settings. In this case the Targeting Expert automatically modifies the names of the subdirectories.

Later, after you have built a Simulator and a Validator for the `component`, for example, then the sub-directory structure was extended to:

```
<target_dir>
+ application._0
+ node._1
+ component._2
+ Simulator
+ Validator
```

Flat Directory Structure

When using an `.sdt` or `.pr` file as input the directory structure will be flattened automatically. I.e it will look like

```
<target_dir>
+ application._0
+ Simulator
+ Validator
```

This is possible because there can only be one node with one component in both cases.

Generated Makefile

Before the selected make tool is invoked the Targeting Expert generates a makefile with all of the necessary settings done. (Please see [“Configure Compiler, Linker and Make”](#) on page 2856 for information on how to set up the makefile generation.)

Caution!

The standard header files of the used SDL to C compiler library are **not** used as dependencies in the compilation rules, i.e. after modifying one of these files you have to select the *Clean* entry in the Make Menu.

Comparisons

There are a few differences that need to be mentioned when comparing the way the Targeting Expert handles makefiles with the way the Organizer's Make dialog does.

When using the Targeting Expert ...

- ... there is no makefile (`systemname.m`) generated by the SDL to C compiler. Instead it generates a sub-makefile (`component_gen.m`) which is automatically parsed and inserted into the Targeting Expert makefile, i.e. the Targeting Expert still gets all the information from the SDL to C compiler but in a different way.
- ... there is not support of template makefiles by the Targeting Expert. The makefiles generated by the Targeting Expert can be modified directly in the so called "user sections". Please see "User Modifications" on page 2924.
- ... there are no `makeoptions/make.opt` files (taken from the kernel directories) used at all. Instead of this, the Targeting Expert gets all the make information from the Pre-defined Integration Settings and places them into its makefile.
- ... the `comp.opt` files placed in the kernel directories are not used at all! Again all the information is taken from the Pre-defined Integration Settings.
- ... the `sccd` cannot be used for pre-processing purposes as default because the Targeting Expert does not use `sccd.cfg` files from the kernel directories. It is recommended to use the Targeting Expert Preprocessor instead.

Note:

The only exceptions in which the Targeting Expert uses some information from the associated kernel directory are the integrations Validation and TTCN Link. In both cases a library is taken from the kernel directory.

User Modifications

After all it is possible that there are a few adaptations to be done, e.g. if there is an assembler file which needs to be assembled and linked.

Example 479 shows the section of a generated makefile where it is possible to add further object files.

Example 479: Generated makefile (1)

```
##### UserObjectsStart (Do not edit this line!) #####
userOBJECTFILES1 =

userOBJECTFILES2 =

##### UserObjectsEnd (Do not edit this line!) #####
```

The list `userOBJECTFILES1` will be inserted at the front of all the object files to link. Accordingly `userOBJECTFILES2` will be inserted at the end.

The rules and the dependencies for the files in Example 479 must be given in the section shown in Example 480.

Example 480: Generated makefile (2)

```
##### UserRulesStart (Do not edit this line!) #####
##### UserRulesEnd (Do not edit this line!) #####
```

Caution!

Any modification that is done outside of the sections described in Example 479 and Example 480 will be overwritten when the makefile is generated anew.

Parameter File `sdtttaex.par`

General

One of the delivered configuration files used by the Targeting Expert is called `sdtttaex.par`. Several different sections give information that is statically used during runtime. Under normal circumstances this file does not need to be modified. However, there might be the need to extend the Targeting Expert functions.

Please see the following sub-sections for more information what can be configured:

- [Compiler Error Descriptions](#)
- [Preprocessor Commands](#)
- [Make Applications](#)
- [C++ Options](#)
- [Debug Options](#)
- [Compiler Dependent Defaults](#)
- [Restricted Compilers](#)
- [Editor Commands](#)
- [Additional Files](#)
- [Host Configuration Options](#) (used for Cmicro only)

Search Order for `sdtttaex.par`

The following shown search order applies when the Targeting Expert is started. The distributed version of `sdtttaex.par` is always placed in `<installationdir>/bin/<platform>bin`

procedure Search_order_on_UNIX

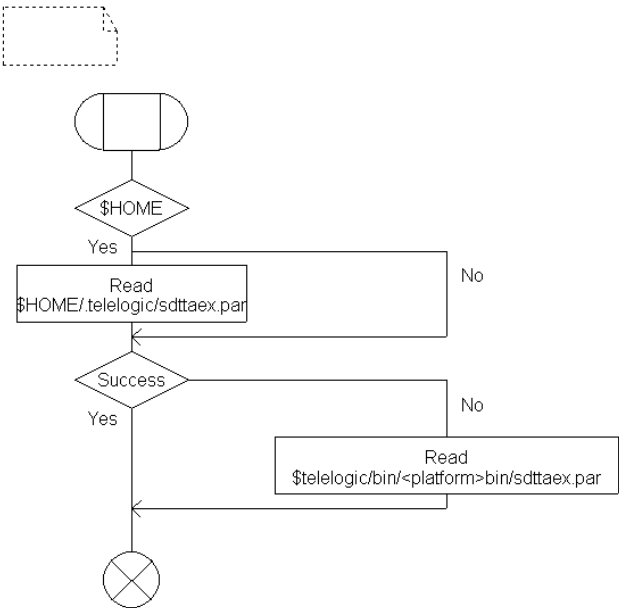


Figure 542: Search order for sdtttaex.par on UNIX

procedure Search_order_in_Windows

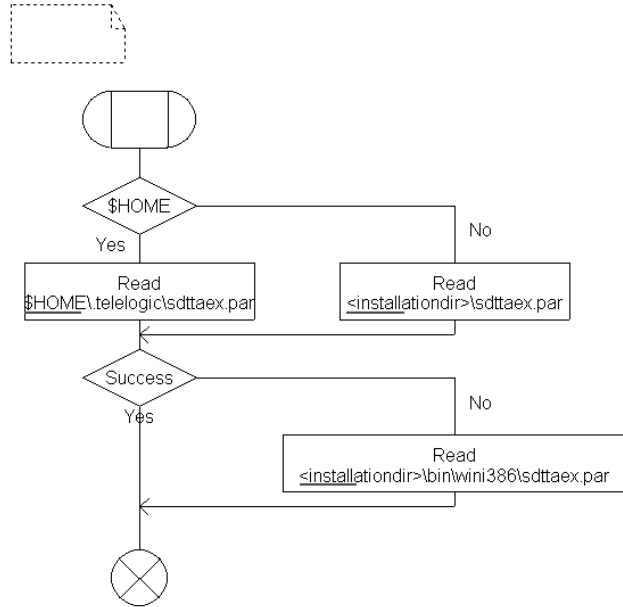


Figure 543: Search order for sdttax.par in Windows

Compiler Error Descriptions

The section [COMPILER-ERROR-DESCRIPTION] of sdttax.par is filled with regular expressions describing the construction of compiler error messages.

Example 481: [COMPILER-ERROR-DESCRIPTION]

[COMPILER-ERROR-DESCRIPTION]				
{cl}	{1}	{.*([1234567890])}	{0}	{2}
{bcc32}	{1}	{Error.*[1234567890]}	{6}	{2}
{cc166}	{2}	{.*[A-Za-z]:}	{0}	{1}
{gcc}	{1}	{.*:[1234567890]}	{0}	{2}
{icc8051}	{1}	{.*,[1234567890]}	{1}	{3}
{c51}	{1}	{OF.*[CH]:}	{3}	{2}
{cc}	{1}	{.*,l}	{1}	{4}
{CC}	{1}	{.*,l}	{1}	{2}
{aCC}	{1}	{:.*,}	{3}	{2}
{ccsimp}	{1}	{.*:[1234567890]}	{0}	{2}
{icpp}	{1}	{!E.*(}	{3}	{1}

Each line gives the description for one compiler. The eight entries have the meaning shown below (explained on the gcc entry):

1. The name of the compiler executable
2. The amount of lines the compiler error message takes
3. The regular expression to determine the erroneous file
4. The amount of characters thrown away at the beginning of the found file name
5. The amount of characters thrown away at the end of the found file name. (In this example the colon and the numeric character do not belong to the file name but both are needed to find it.)
6. The regular expression to determine the line number
7. The amount of characters thrown away at the beginning of the found line. (In this example the colon at the front of the regular expression does not belong to the line number but is needed to find it).
8. The amount of characters thrown away at the end of the found line. (In this example the colon at the end of the regular expression does not belong to the line number but is needed to find it).

Preprocessor Commands

This section takes the preprocessor commands for all the default compilers that can be set in the preferences. It has got the following contents:

Example 482: [PREPROCESSOR-COMMANDS]

```
[PREPROCESSOR-COMMANDS]
{Sun GNU gcc}      {gcc -P -E -C}
{Sun Workshop cc}  {cc -C -P}
{Sun Workshop CC}  {CC -E}
{HP GNU gcc}       {gcc -P -E -C}
{HP cc}            {cc -C -P}
{HP aCC}           {aCC -E}
{Microsoft}        {cl -P -EP -C -nologo}
{Borland}          {cpp32 -P-}
```

Make Applications

The section [MAKE_APPLICATIONS] of `sdttaex.par` is used to specify the command line of the make application, i.e. how it will be invoked.

```
[MAKE APPLICATIONS]
{Borland make}
{Borland make (ignore exit codes)}
{Borland make (using temporary response file)}
{Microsoft nmake}
{Microsoft nmake (ignore exit codes)}
{Microsoft nmake (using temporary response file)}
{Tasking mk166}
{Tornado make}
{UNIX make}
```

1. The text to identify it in the make configuration. See [“Make tool” on page 2869](#)
2. The name of the make application and the option used to specify the makefile.
3. This is the start sequence for the generation of temporary response files. Empty for most make applications.
4. This is the stop sequence for the generation of temporary response files. Empty for most make applications.

The C++ options specify what has to be added to the compiler's Options and the linker's Options to compile and link the SDL to C compiler's library as C++ code.

Example 484: [C++-OPTIONS]

[C++OPTIONS]		
{Sun GNU gcc}	{-xc++}	{-lstdc++}
{HP GNU gcc}	{-xc++}	{-lstdc++}
{Microsoft}	{-TP}	
{Borland}	{-P}	

1. compiler name
2. compiler command line option
3. linker command line option

Debug Options

The debug options specify what has to be added to the compiler's Options and the linker's Options to compile and link the target application including debug information.

Example 485: [DEBUG_OPTIONS]

[DEBUG-OPTIONS]		
{Sun GNU gcc}	{-g}	{-g}
{Sun Workshop cc}	{-g}	{-g}
{Sun Workshop CC}	{-g}	{-g}
{HP GNU gcc}	{-g}	{-g}
{HP cc}	{-g}	{-g}
{HP aCC}	{-g}	{-g}
{Microsoft}	-D "_DEBUG"	/debug
{Borland}	-v	-v
{icc12}	-g	-g

Each line consist of the three entries

1. compiler name
2. compiler command line option
3. linker command line option

Compiler Dependent Defaults

The compiler dependent defaults ease the implementation of new pre-defined integrations because the values given here do not need to appear again in the .its file(s).

Example 486: [COMPILER-DEPENDENT-DEFAULTS]

```
[COMPILER-DEPENDENT-DEFAULTS]
{Borland} {bcc32} {bcc32} {tlib} {.obj} {.exe} {.lib} {%I -c -w- -DUSING_DLL -
DIC86 -o%o %s} {-e%e %L %O} {%l /C /E /P256 %O} {-I$(sctCODERDIR)} {bcc_obj}
{Borland make (using temporary response file)}
```

The different entries mean the following (listed from left to right):

1. Compiler name
2. Compiler tool name
3. Linker tool name
4. Library manager name
5. Object file extension
6. Executable file extension
7. Library file extension

8. Compiler options
9. Linker options
10. Library Manager Options
11. Coder include path
12. Relative path to intermediate directory
13. make application to use per default

Restricted Compilers

There are a few target compilers which do not support file names which are longer than 8.3 (8 character in front of '.' and a 3 character extension). To ensure that files are generated only which fit to the 8.3 requirement. This section looks as follows:

Example 487: [8.3-COMPILERS]

```
[8.3-COMPILERS]
{c51}
{icpp}
{icc12}
```

Editor Commands

The Targeting Expert allows to use other text editor different to the build-in one. The information how to start these editors is given in the section [EDITOR-COMMANDS] shown below.

Example 488: [EDITOR-COMMANDS]

```
[EDITOR-COMMANDS]
{Windows} {TextPad}           {textpad -ca -q "%f" (%1,0) }
{Windows} {UltraEdit}        {UEdit32 %f/%1/0}
{Windows} {Codewright}       {codewright %f -G%1}
{Windows} {MS DeveloperStudio} {msdev %f}
{UNIX}    {Emacs}            {emacs %f}
{UNIX}    {dtpad}            {/usr/dt/bin/dtpad %f}
{UNIX}    {nedit}            {nedit -line %1 %f}
```

There are 2 placeholders used in the command. %f will be replaced by the name of the file to open and %1 gives the line number which should be shown.

Additional Files

The sections shown in [Example 489](#) will be used to find out which files belong to external parts of the distributed SDL to C compiler libraries.

Example 489: Additional Files Sections

```

[CMICRO-KERNEL-FILES]
$(sctkernelldir)/mk_main.c
$(sctkernelldir)/mk_sche.c
$(sctkernelldir)/mk_queue.c
$(sctkernelldir)/mk_outp.c
$(sctkernelldir)/mk_timl.c
$(sctkernelldir)/sctpred.c

[CMICRO-TI-KERNEL-FILES]
$(sctkernelldir)/ti_sche.c
$(sctkernelldir)/ti_queue.c
$(sctkernelldir)/ti_outp.c
$(sctkernelldir)/ti_timl.c
$(sctkernelldir)/ti_init.c
$(sctkernelldir)/sctpred.c

[TESTER-FILES]
$(scttesterldir)/mt_tsdl.c
$(scttesterldir)/mt_tsys.c
$(scttesterldir)/mt_cod.c
$(scttesterldir)/mt_cmd.c
$(scttesterldir)/mt_opt.c
$(scttesterldir)/mt_deb.c
$(scttesterldir)/mt_rec.c
$(scttesterldir)/mt_play.c
$(sctkernelldir)/ml_buf.c

[TI-TESTER-FILES]
$(scttesterldir)/mt_tsdl.c
$(scttesterldir)/mt_tsys.c
$(scttesterldir)/mt_cod.c
$(scttesterldir)/mt_cmd.c
$(scttesterldir)/mt_opt.c
$(scttesterldir)/mt_deb.c
$(sctkernelldir)/ml_buf.c

[CODER-FILES]
$(sctCODERDIR)/cucf_er.c
$(sctCODERDIR)/cucf_er_sdt.c
$(sctCODERDIR)/bms/bms.c
$(sctCODERDIR)/bms/bms_small.c
$(sctCODERDIR)/ems/ems.c
$(sctCODERDIR)/ems/ems_eo_sdt.c
$(sctCODERDIR)/er/ascii/ascii.c
$(sctCODERDIR)/er/ber/ber_base.c
$(sctCODERDIR)/er/ber/ber_content.c
$(sctCODERDIR)/er/ber/ber_decode.c
$(sctCODERDIR)/er/ber/ber_encode.c
$(sctCODERDIR)/er/per/per_base.c
$(sctCODERDIR)/er/per/per_content.c
$(sctCODERDIR)/er/per/per_decode.c
$(sctCODERDIR)/er/per/per_encode.c
$(sctCODERDIR)/er/mms/mms.c

```

```
$(sctCODERDIR)/vms/vms_base.c
$(sctCODERDIR)/vms/vms_check.c
$(sctCODERDIR)/vms/vms_export.c
$(sctCODERDIR)/vms/vms_print.c

[TCP-IP-FILES]
$(sctTCPIPDIR)/tcpipcomm.c
```

Note:

The [TESTER-FILES] and [TI-TESTER-FILES] only apply for the Cmicro SDL to C compiler.

The [TCP-IP-FILES] only applies for the Cadvanced SDL to C compiler.

Host Configuration Options

The host configuration options are the default host configurations for different compilers when using the Target Tester, i.e. these options are only used for Cmicro applications. An example can be found below.

Example 490: [HOST-CONFIGURATION-OPTIONS] (Borland compiler)

```
[HOST-CONFIGURATION-OPTIONS]
bcc32 "UNIT-NAME                sec"
bcc32 "UNIT-SCALE                1.0"
bcc32 "LENGTH_CHAR              1"
bcc32 "LENGTH_SHORT             2"
bcc32 "LENGTH_INT               4"
bcc32 "LENGTH_LONG              4"
bcc32 "LENGTH_FLOAT             4"
bcc32 "LENGTH_DOUBLE            8"
bcc32 "LENGTH_POINTER           4"
bcc32 "ALIGN_CHAR               8"
bcc32 "ALIGN_SHORT              8"
bcc32 "ALIGN_INT                8"
bcc32 "ALIGN_LONG               8"
bcc32 "ALIGN_FLOAT              8"
bcc32 "ALIGN_DOUBLE             8"
bcc32 "ALIGN_POINTER            8"
bcc32 "ENDIAN_CHAR              1"
bcc32 "ENDIAN_SHORT             21"
bcc32 "ENDIAN_INT               41"
bcc32 "ENDIAN_LONG              41"
bcc32 "ENDIAN_FLOAT             41"
bcc32 "ENDIAN_DOUBLE            81"
bcc32 "ENDIAN_POINTER           41"
```

External Makefile Generator

General

As it is probably necessary for you to modify the layout of the generated makefile, the source code of the external makefile generator “makegen” is delivered with every Telelogic Tau distribution. It can be found in `<installationdir>/sdt/sdtdir/util/<platform>.`

Source and Make Files

The source files delivered are:

- `makegen.[ch]`
- `ini_api.h`
- `pdm_api.h`

Furthermore there are makefiles and libraries:

- **cc - compiler on UNIX**
 - `makegengcc.m`
 - `makegenlib.a`
- **gcc 2.95.2 - compiler on UNIX**
 - `makegengcc.m`
 - `makegenlib.a`
- **Microsoft VC++ 6.0 compiler in Windows**
 - `makegencl.m`
 - `makegencl.lib`
- **Borland C++ Builder 5.51 compiler in Windows**
 - `makegenbcb.m`
 - `makegenbcb.lib`

Utilities

General

Delivered in combination with the Targeting Expert there is an application called `taexutil`. This application can be used directly from the Targeting Expert *Tools* menu or on the command line, e.g. in makefiles.

The following utility functions are supported:

- DOS to UNIX
- UNIX to DOS
- Indent
- Preprocessor

Note:

Each utility will create a backup file called `<inputfile>.bak`

DOS to UNIX

This utility can be used to modify ASCII files. It replaces all the found `'\r\n'` sequences against `'\n'`.

- Command line:
`taexutil D2U <inputfile> <outputfile>`
`<inputfile>` and `<outputfile>` can be equal but both must be given.

UNIX to DOS

This utility can be used to modify ASCII files. It replaces all the found `'\n'` characters against `'\r\n'` sequences.

- Command line:
`taexutil U2D <inputfile> <outputfile>`
`<inputfile>` and `<outputfile>` can be equal but both must be given.

Indent

The indentation of the given ASCII file will be corrected. This means

- all the preprocessor directives (starting with `'#'`) will be moved into the first column. This is necessary to conform with K&R compilers.

- all the other lines will be indented according to the given blocks in C, i.e. it is controlled by the use of '{' and '}' characters.
- all TAB characters ('\t') will be removed.
- all the SPACE (' ') characters at the end of lines will be removed.

Caution!

The indent offered by here does not offer the same functionality as indent known from UNIX!

- Command line:
taexutil INDENT <inputfile> <outputfile>
<inputfile> and <outputfile> can be equal but both must be given.

Preprocessor

Note:

The main intention of the preprocessor given here is **not** to get an easy to read source file. Instead it is to get a preprocessed file that can be compiled by a target compiler (e.g. Keil, Tasking, ...) because long macros are not supported by some target compilers.

The preprocessor utility should only be used to preprocess generated C files. The following tasks will be processed:

1. Copy the SDL to C compilers main header file (ml_typ.h/sct-types.h) beside the output file.
2. Put all the lines of the main header file containing `#include <...>` statements into comments
3. Preprocess the selected generated C file by using the default compiler (see Telelogic Tau Preferences).
4. Remove all the empty lines in the preprocessed file.
5. Run Indent on the preprocessed file.
6. Remove all the comments surrounding `#include < ...>` in the preprocessed file.
7. Remove the private copy of the main header file.

Note:

All the files given with `#include "..."` will be included during the preprocessing.

- Command line:
`taexutil PREPRO <inputfile> <outputfile> "<cmd>"`
`<compiler_flag>`
 - `<inputfile>` and `<outputfile>` can be equal but both must be given.
 - `<cmd>` is the command line needed to invoke the preprocessor of the default compiler
 - `<compiler_flag>` is the define to select the target compiler.
(Please see [“Compiler Flag”](#) on page 2862)

FAQs

- **How can I set the target directory in the Targeting Expert?**

It is not possible to set the target directory in the Targeting Expert. The one given in the Organizer will be used instead. To make it possible to have different integrations in one target directory the Targeting Expert creates a Target Sub-Directory Structure.

- **Where can I add my own macro definitions?**

There is a manual configuration file written for each integration. It is called `sct_mcf.h` for Cadvanced and `ml_mcf.h` for Cmicro.

This header file is included in the SDL to C compiler's library during compilation and contains a section where you can insert your own macro definitions. The section begins with

```
/* BEGIN User Code */
```

and ends with the line

```
/* END User Code */
```

All the text given between both lines will be saved if the manual configuration file is re-generated.

Caution!

The manual configuration file `sct_mcf.h` for Cadvanced is included only if the flag `USER_CONFIG` is set. This can be done in most cases by defining `-DUSER_CONFIG` in the compiler options.

- **How can I modify the value of `$sdt_dir` in the generated make-file?**

Per default the value of `$sdt_dir` is set to

`<installationdir>/sdt/sdt_dir/<platformsdt_dir>`. In some cases it is probably necessary to get the library files from another directory. In this case you can modify the *Library Directory* in the Make dialog. The sub-directory structure must be the same as given under `<installationdir>/sdt/sdt_dir/<platformsdt_dir>`.

- **How can I modify the configuration flags that are disabled in the Targeting Expert user interface?**

The Targeting Expert can be customized in a way so that the configuration flags listed under Configuration Settings to Be Set or Configuration Settings to Be Reset are not disabled. To achieve that open the Customize dialog via the menu choice *Tools > Customize* and select the Advanced Mode. (Please see “Customization” on page 2850 for more details)

- **How can I create my own pre-defined integration?**

You can create your own pre-defined integrations by

- selecting one of the distributed pre-defined integrations (the one that fits most to your needs)
- configure it as you like (maybe it is necessary to switch into the Advanced Mode to have access to all the dialogs)
- export the settings like it is described in “Export” on page 2848#

After you have done so the pre-defined integration will be available.

- **I have exported an pre-defined integration but my colleagues are not able to access it in my home directory/in my PC installation. What to do?**

When exporting settings to pre-defined integration settings it is possible to

- export as project settings - or
- export as user settings

You have probably exported as user settings, i.e. the *.its* file is not accessible for other users. You can move the `<integration>.its` file in the same directory where the `<systemname>.sdt` file of your project is saved. So it will be visible for all users working on this system.

